SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

LEVEL II

1. REPORT NUMBER: HPP80-23	2. GOVT ACCESSION NO.: AD-A096779
4. TITLE (and Subtitle): RLL-1: A Representation Language Language. Supplement: Details of RLL-1.	5. TYPE OF REPORT & PERIOD COVERED: Technical rept.
	6. PERFORMING ORG. REPORT NUMBER: ---
7. AUTHOR(s): Russell Greiner and Douglas B. Lenat	8. CONTRACT OR GRANT NUMBER(s): N00014-80-C-0609
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science Department, Stanford University, Stanford, California 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ---
11. CONTROLLING OFFICE NAME AND ADDRESS: Mathematical & Information Sciences Div. Office of Naval Research, 800 No. Quincy Street, Arlington, Va. 22217	12. REPORT DATE: October 1980
	13. NUMBER OF PAGES: 68
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): ---	15. SECURITY CLASS. (of this report): ---
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: ---

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT A
Approved for public release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Reports Distribution List, July 31, 1980

18. SUPPLEMENTARY NOTES: ---

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Representation, Knowledge, Language, Self-Description, Self-Modification, Expert Systems.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DTIC ELECTE MAR 25 1981 S D

D

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

094120

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

81 2 18 001

AD A096779

DTIC FILE COPY

Stanford Heuristic Programming Project
HPP-80-23 (Working Paper)

October 1980

Details of RLL-1

by

Russell Greiner and Douglas B. Lenat
Computer Science Department
Stanford University

Supplement to
"RLL-1:
A Representation Language Language"

Heuristic Programming Project
Computer Science Department
Stanford University
Stanford, California 94305

Details Of RLL-1

This paper includes many implementation level details about the RLL-1 system, described in a companion paper, "RLL-1: A Representation Language Language" (Heuristic Programming Project Working Paper HPP-80-9, October 1980, at Stanford University, by Russell Greiner).

THE CONTENTS ARE AS FOLLOWS:

Table of Contents

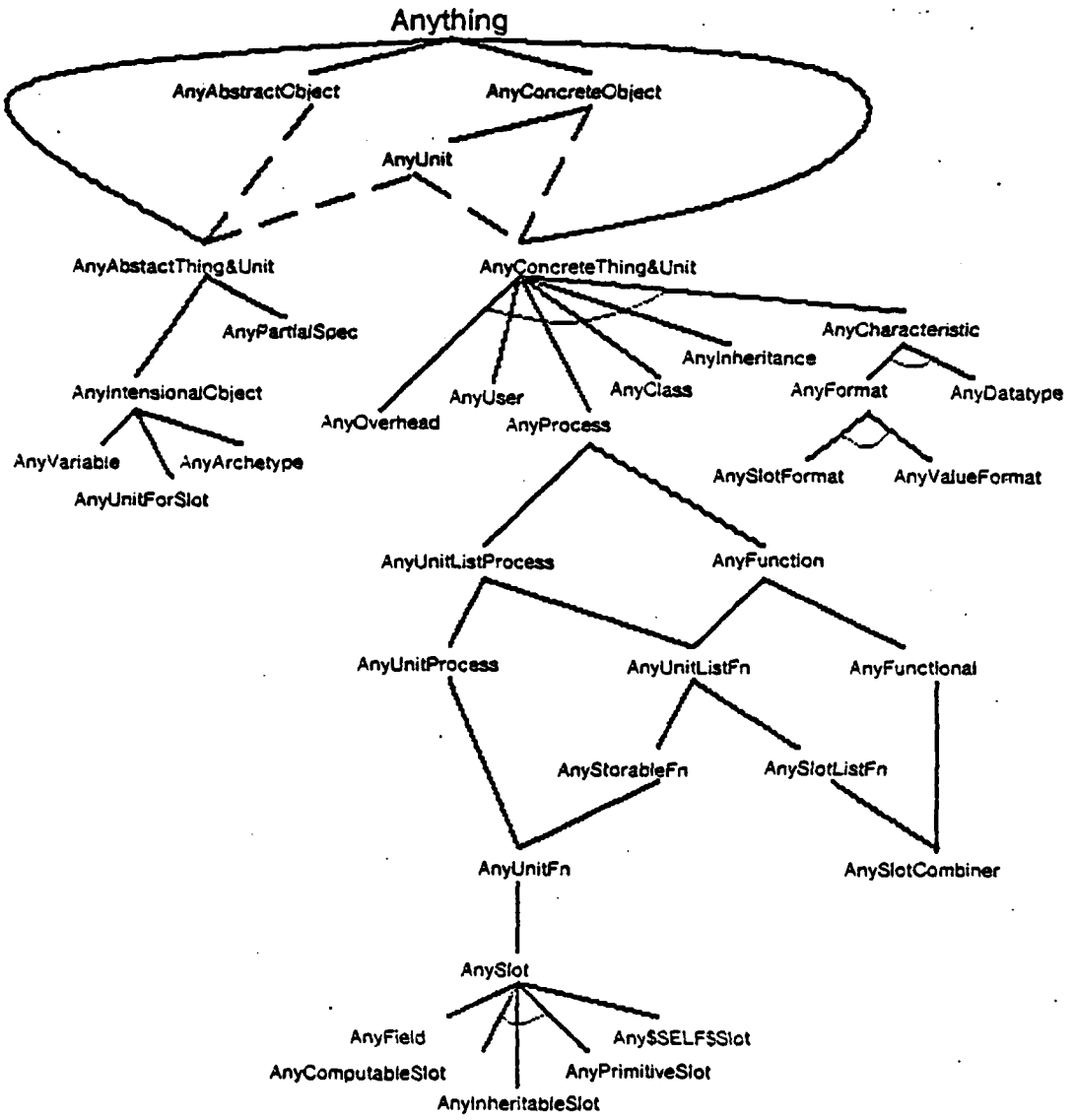

Diagram # 1

Classes of Units

Appendix E: Special Units

Many RLL-1 units are directly used by one or more of the RLL-1 functions listed below. These special ones are enumerated below, following a depth first traversal of the RLL-1 Knowledge Base. Diagram #1 portrays a skeleton of this hierarchy, showing the subset relations joining these various classes.

E.1: Naming Conventions.

Any***	- refers to the class of all *** objects [e.g. AnySlot refers to *slots*]
Typical***	- refers to the abstract object which typifies members of Any***
F***	- refers to a format [e.g. FSingleton]
I***	- refers to inheritance type [e.g. IExamples]
My***	- is a syntactic slot, [e.g. MyCreator]
***-Instances	- holds value of syntactic slot, My***, to be inherited.
	- This ***-Instances slot appears in Typical— units
All***s	- refers to extension of *** slot [e.g. AllIsas extends Isa]
***Type	- refers to a datatype, [e.g. IntegerType]
FnFor***	- the value of this slot is a function, [e.g. ForForGetting]
To***	- the value of this slot is a function, ...
***Value	- the value of this slot refers to slot (as opposed to a field)
***Field	- the value of this slot refers to field (currently not used)

E.2: Legend.

This RLL-1 Knowledge Base is organized into classes, which each contain a set of elements. Associated with each such class of units is a list of slots, which are meaningful for elements of this class. For example, the *Datatype* slot makes sense for any function; but is meaningless for, say, people.

Each entry is Section E.3 will describe a class of units; including relevant features of the members of this class, and the list of slots defined for each member. The format used will be:

Class Name (*j*) - A short description of what the class Class Name represents.
SuperClass: {Important immediate supersets of this class}
SubClass: {Important immediate subsets of this class}
There are currently *n* examples.

Direct Examples: {Important examples of this class}

The following slots are defined for all "members of Class Name"s:

slot$_1$ – [slot$_1$'s range] A short description of slot$_1$.
Inverse: slot$_1$'s inverse
HLDefn: slot$_1$'s high level definition
ArgList: arg$_1$, arg$_2$, ... arg$_{M_1}$

.
.
.

slot$_N$ – [slot$_N$'s range] A short description of slot$_N$.
Inverse: slot$_N$'s inverse
HLDefn: slot$_N$'s high level definition
ArgList: arg$_1$, arg$_2$, ... arg$_{M_N}$

We view a slot as a function, which maps a unit onto some value. The "slot$_i$'s range" field above encodes the space of permissible values the *slot$_i$* of any unit may take. There are several basic categories of slots, each with its own type of range. When *Range is FunctionType*, it is relevant to know what slot$_i$'s *HighLevelDefinition*, and *ArgList* is – otherwise these slots are not even well defined. Similarly, unless *Range* is UnitType, it is not possible for Slot$_i$ to have an inverse.

Subsection E.4 will provide an index to the units presented in Section E.3, using the sequential numbers assigned to each class. Each class will refer to that number, and each slot will point to the number associated with the class to which is affiliated.

E.3: Actual Units.

Anything (*1*) - The topmost node, in all the hierarchies.
SubClass: AnyAT&U AnyAbstractThing AnyCT&U AnyConcreteThing AnyFocus AnyUnit
Direct Examples: AnyClassOfObjects

The following slots are defined for all "thing"s:

Isa – [FSet (UnitType (*P AnyClassOfObjects))] This primitive slot is the fundamental hierarchical link in this system, specifying those classes to which this unit belongs. Note that its format is SET - hence this system can handle a DAG structure; better for our purposes than a tree.
Inverse: UnitExamples

Characteristics – [FSet NonNILType] This lists some essential characteristics of this unit. (Not currently in use.)

HighLevelDefn: (Unioning FunctionCharacter FormatCharacter)

Descr – [FSingleton NonNILType] This describes this unit. (It is used to generate this document.)

AllIsas – [FSet UnitType] This specifies each class to which this unit belongs. (It includes each *SuperClass* of this unit's *Isa* slot.)
Inverse: AllExamples
HighLevelDefn: (Composition SuperClass* Isa)

AllGenls – [FSet UnitType] This points to a list of those units which are somehow more general than this unit.
Inverse: AllSpecs
HighLevelDefn: (Unioning Prototypes (OneOf SuperClass SuperTypEx* SuperSlot* GenlAct*))*

AllSpecs – [FSet UnitType] This lists every unit which is somehow more precise than this unit.
Inverse: AllGenls
HighLevelDefn: (Unioning AllTypicalExampleOfs (OneOf SubClass* SubTypEx* SubSlot* SpecAct*))

Prototypes – [FSet (UnitType (*P AnyArchetype))] This points to each typical example of this unit, not necessarily in order of increasing generality.
Inverse: AllTypicalExampleOfs
HighLevelDefn: (Composition TypicalExample *AllIsas*)

Specializations – [FSet UnitType] This points to each unit which "specialises" this unit.
Inverse: Generalizations
HighLevelDefn: (OneOf SubSlot SpecAct SubTypEx SubDT SubClass)

OrderedPrototypes – [FOrderedSet UnitType] Enumerates the prototypes of this unit in order of increasing generality (i.e. TypicalDog would precede TypicalAnimal .)
HighLevelDefn: (PutInOrder Prototypes SuperTypEx* NIL MembForOrdPro)

AnyAT&U (2) - This is a HACK - to deal with the units in this system, which represent both some abstract object (NOT in the world,) and themselves...
SuperClass: Anything
SubClass: AnyIntensionalObject AnyPartialSpec

The slots appropriate for all "AT&U"s are those defined for each of: (AnyUnit AnyAbstractThing *).*

AnyIntensionalObject (3) - Descendants of this unit describe some entity in the world intensionally – as opposed to directly referring to in.
SuperClass: AnyAT&U AnyCT&U
SubClass: AnyArchetype AnyDescriptor AnyUnitForSlot AnyVariable

AnyArchetype (4) - Every typical example of some class is an archetype, and descends from this unit.

SuperClass: AnyIntensionalObject

There are currently 123 examples.

The following slots are defined for all "Archetype"s:

NewPossibleSlots – [FSet SlotType] This lists the names of slots which are meaningful for every "instance" of this typical example. Furthermore, this is the highest place where this slot is meaningful (hence the "newness" of the name.)

Inverse: MakesSenseFor

HighLevelDefn: (Composition (Soften NewPossibleSlots) MyComposeOf)

TypicalExampleOf – [FSingleton (UnitType (*P AnyClassOfObjects))] Refers back, from the typical example unit, to the class of elements it typifies.

Inverse: TypicalExample

SuperTypEx – [FSet UnitType] This denotes the relation connecting TypicalDog to TypicalAnimal – i.e. a superset relation holds between the elements each (respectively) typifies.

Inverse: SubTypEx

HighLevelDefn: (Composition TypicalExample SuperClass TypicalExampleOf)

SubTypEx – [FSet UnitType] See *SuperTypEx* .

Inverse: SuperTypEx

HighLevelDefn: (Composition TypicalExample SubClass TypicalExampleOf)

PossibleSlots – [FSet UnitType] This is obsolete - and will soon be deleted.

HighLevelDefn: (Composition NewPossibleSlots SuperTypEx*)

*SuperTypEx** – [FSet UnitType] This is the transitive closure of *SuperTypEx* .

Inverse: SubTypEx*

HighLevelDefn: (Composition TypicalExample SuperClass* TypicalExampleOf)

*SubTypEx** – [FSet (UnitType (*P AnyArchetype))] This is the transitive closure of *SubTypEx* .

Inverse: SuperTypEx*

HighLevelDefn: (Composition TypicalExample SubClass* TypicalExampleOf)

AnyDescriptor (5) - This will eventually hold descriptors – a whole class of entities which will have to be defined...

SuperClass: AnyIntensionalObject

AnyUnitForSlot (6) - At times, there is more than just one "morsel" of information needed to describe the value of some unit's slot. RLL then devotes an entire unit to hold this information. Such units descend from this AnyUnitForSlot.

SuperClass: AnyIntensionalObject

The following slots are defined for all "UnitForSlot"s:

vaLue – [FSingleton NonNILType] When a unit is allocated to store facts about the value of a slot, the actual value of that slot, if any, is kept in the **vaLue** slot of that sub unit.

HighLevelDefn: (Application (Composition Defn LivesInSlot) LivesInUnit)

AnyVariable (7) - This class contains the universally or existentially bound variables. Note that this is a META description of said units.

SuperClass: AnyIntensionalObject

AnyPartialSpec (8) - This class includes objects which are only partial specified. This is essential to deal with MOLGEN UNITs package notion of SPEC inheritance – in which some object is specified more and more completely.

SuperClass: AnyAT&U

SubClass: AnyGenericEvent

The following slots are defined for all "PartialSpec"s:

MyRefineSlots – [FSet SlotType] The value of U:*MyRefineSlots* is a list of those slots on the unit U which are used to specify facts which are not definitional.

AnyAbstractThing (9) - Instances refer to intangible objects; as opposed to concrete things (such as real world people or units).

SuperClass: Anything

AnyCT&U (10) - This is a HACK - to deal with the units in this system, which represent both some object in the world, and themselves...

SuperClass: Anything

SubClass: AnyCharacteristic AnyClassOfObjects AnyDecomposableObject AnyEvent AnyInheritance AnyIntensionalObject AnyOverhead AnyProcess AnyUnit AnyUser

The slots appropriate for all "CT&U"s are those defined for each of: (AnyUnit AnyConcreteThing *).*

AnyCharacteristic (11) - This fathers units which describe characteristics of some entity – as opposed to something which actually exists in and of itself.

SuperClass: AnyCT&U

SubClass: AnyDatatype AnyFormat

AnyDatatype (*12*) - Every datatype (used for building up type specifications,) descends from this.

SuperClass: AnyCharacteristic

Direct Examples: KBType NonNILType NumberType BooleanType UnrestrictedType FunctionType SlotType IntegerType UnitType StringType

The following slots are defined for all 'Datatype"s:

EqualDTSpec – [FSet (FListN (UnitType (*P AnyDatatype)) UnrestrictedType)] This helps relate one datatype to another – by defining a set of equivalent datatype-datarange pairs.

VerifyType – [FSingleton FunctionType] DT:*VerifyType* is a predicate, returning *nonNIL* for all elements which qualify in this datatype.

GenerateAll – [FSingleton FunctionType] DT:*GenerateAll* returns a list of all members of the datatype, DT .

SuperDT – [FSet (UnitType (*P AnyDatatype))] This points to the list of more general datatypes – i.e. those which contain a superset of this datatype's members.

Inverse: SubDT

SubDT – [FSet (UnitType (*P AnyDatatype))] DT:*SubDt* points to Datatypes which accept a subset of those accepts by the datatype DT .

Inverse: SuperDT

RangeInterpreter – [FSingleton FunctionType] DT:*RangeInterpreter* is a function which helps parse the value of S:*DataRange* , which is used to determine the appropriate values to fill U:*S* .

IsTypeOf – [FSet SlotType] Points from a datatype to those units representing functions whose range is composed of this datatype.

Inverse: Datatype

*SuperDT** – [FSet UnitType] A list of a unit's SuperDT, THEIR SuperDT, etc.

Inverse: SubDT*

HighLevelDefn: (Starring SuperDT)

*SubDT** – [FSet (UnitType (*P AnyDatatype))] Transitive closure of SubDT .

Inverse: SuperDT*

HighLevelDefn: (Starring SubDT)

AnyFormat (*13*) - From this descends the units which each serve to describe some format; which can be used in a type description.

SuperClass: AnyCharacteristic

SubClass: AnySlotFormat AnyValueFormat

The following slots are defined for all 'Format"s:

FnForAdding – [FSingleton FunctionType] One should add a new entry to a Ordered LIST in a different manner than one uses to add a value to an Unordered SET. (In the first case, multiple occurances of an element are acceptable; which is NOT true in the second case.) This information is contained in the function stored in F:*FnForAdding* . Note it is used by the various *Add* ing functions – such as the one employed to add a new entry to the existing value of a unit's slot.

FnForDeleting – [FSingleton FunctionType] There are different ways of deleting an element from a list, versus from a set. As with *FnForAdding* , this information is stored in F:*FnForDeleting* , where F is the name of a format.

FormatCharacter – [FSet NonNILType] This holds a list of specifications for some format. It is not currently used for anything but show.

FnForGetting – [FSingleton FunctionType] This is used to determine the value of a special slot value, to be returned when *GetValue* requests its value.

FnForPutting – [FSingleton FunctionType] This indicates how to put a value. It is basically used for indirect pointers.

FnForKilling – [FSingleton FunctionType] To handle indirect pointers using the **Do** special slot values, one needs assistance to describe how to perform various manipulations – such as deleting a full slot's value. This information is kept in VF:*FnForKilling* , where VF is a value format.

FnForVerifyingAll – [FSingleton FunctionType] This is used, in conjunction with the various verifying functions associated with each datatype, to build the function stored in the *VerifyValue* slot of each slot. It indicates how to verify that a full entry is correct, given, (among other arguments,) the predicate to apply to each entry individually.

FnForVerifyingElement – [FSingleton FunctionType] This is used to verify that a single element is correct. (Perhaps this isn't used anymore – I must look into this.)

AnySlotFormat (*14*) - Descendants are used in type specifications, for processes (which include functions and slots) .

SuperClass: AnyFormat

Direct Examples: FSingleton FList FSet FOrderedSet FBag FListN

AnyValueFormat (*15*) - Descendants are used in a **Do** special slot value format. This has applications as indirect pointers, as well as to put epistemological marks on some value.

SuperClass: AnyFormat

SubClass: AnyIndirectPtrFormat

Direct Examples: FOneOf FExecute

AnyIndirectPtrFormat (*16*) - These are used to deal with subunits, and other places where much data is stored at another location – other than U:*S* .

SuperClass: AnyValueFormat

Direct Examples: FSeeUnit FSeeSlot FSeeU&S

AnyClassOfObjects (*17*) - Every member of this class is itself a set.

SuperClass: AnyCT&U

There are currently 47 examples.

The following slots are defined for all "ClassOfObjects"s:

TypicalExample – [FSingleton (UnitType (*P AnyArchetype))] This points from a class to an abstract entity which holds default information about members of this class.

Inverse: TypicalExampleOf

DomainOf – [FSet (UnitType (*P AnyFunction))] If a function takes one or more elements of this class, as arguments, that function is stored on the *DomainOf* slot of the unit representing that class.

Inverse: Domain

RangeOf – [FSet (UnitType (*P AnyFunction))] If a function maps into this class (or a space having this class as one dimension,) that function is stored on the *RangeOf* slot of the unit representing that class.

Inverse: Range

IntensionalExamples – [FSet UnitType] These examples are all intentional objects – that is, they are only defined intentionally.

StdExamples – [FSet UnitType] This slot basically represents the vanilla "$\in$" relationship, between an extensional object, and a represented set.

SuperClass – [FSet (UnitType (*P AnyClassOfObjects))] This points from a class, C, to each superset of C, D_i . That is, $x \in C \Rightarrow x \in D_i$, for all elements x, and all sets, D_i .

Inverse: SubClass

SubClass – [FSet UnitType] This points from a class, C, to each subset of C.

Inverse: SuperClass

UnitExamples – [FSet UnitType] This points from a class to each member of that class. (Both constant and variable.)

Inverse: Isa

HighLevelDefn: (Unioning IntensionalExamples StdExamples)

TotalSoFar – [FSingleton IntegerType] Fill in later.

*SuperClass** – [FSet UnitType] A list of a unit's SuperClass, THEIR SuperClass, etc.

Inverse: SubClass*
HighLevelDefn: (Starring SuperClass)

*SubClass** – [FSet UnitType] A list of a unit's SubClass, THEIR SubClass, etc.
Inverse: SuperClass*
HighLevelDefn: (Starring SubClass)

AllExamples – [FSet UnitType] This points from a class to a list of members of this class. (*UnitExamples* only pointed to elements immediately a member of some set — this will follow their *SuperClass* links as well, to more accurately represent an "∈" relation.)
Inverse: AllIsas
HighLevelDefn: (Composition UnitExamples SubClass*)

GenlsModels – [FSet (UnitType (*P AnyArchetype))] This is used for several of the inheritances. The prototypes of every example of some class include that class's *GenlsModels* .
HighLevelDefn: (Composition TypicalExample SuperClass*)

AnyDecomposableObject (*18*) - Descendants of this are real world entities which consist of several subparts; and which are little more than the union of such pieces.
SuperClass: AnyCT&U
SubClass: AnyActionSequence

The following slots are defined for all "DecomposableObject"s:

ComposedOf – [FSet NonNILType] This points to a list of the parts associated with this entity.

AnyActionSequence (*19*) - Any compound action, composed a sequence of subactions, descends from this unit.
SuperClass: AnyAction AnyDecomposableObject

The following slots are defined for all "ActionSequence"s:

SubActions – [FList NonNILType] Each action may be broken into a series of substeps – each of which is a "subaction".

AnyInheritance (*20*) - All modes of inheritance will descend from here. Associated with each instance of an inheritance is a means for creating new units, and constraints on properties these units may acquire. (and maybe other things...)
SuperClass: AnyCT&U
Direct Examples: IExamples ISubClass ITypEx

The following slots are defined for all "Inheritance"s:

UseToGetSlots – [FSingleton NonNILType] This points to a high level definition of a function which is takes the parent units, and returns a list of units whose *NewPossibleSlots* slot together hold the slots which should be initialized in this new offspring.

GetPossibleSlotsFn – [FSingleton SlotType] This points a unit which represents a function, which takes the units found using the *UseToGetSlots* function mentioned above, and returns a list of values with which to initialise a new unit. Each entry in this ordered list is a triple, consisting of the name of the slot, followed by the location of the relevant initialising function, and the typical example in which this slot was found.

Inverse: UsedByInheritance

AnyOverhead (*21*) - Miscellaneous information needed by CORLL, etc., is stored on units which descend from here.

SuperClass: AnyCT&U

SubClass: AnyStatus

AnyStatus (*22*) - This will father all *.STATUS units

SuperClass: AnyOverhead

Direct Examples: RLL.STATUS

The following slots are defined for all "Status"s:

KBsVARS – [FSingleton NonNILType] This names a variable, whose value lists the variables associated with this Knowledge Base.

KBsConnectedTo – [FSet (UnitType (*P AnyStatus))] The value of <kb>.STATUS:*KBsConnectedT* is a list of other status units, which were resident in core the last time this Knowledge Base, <kb>, was used. It is reset whenever a new Knowledge Base is read in, or whenever one is disconnected or reconnected.

Inverse: KBsConnectedTo

OpenDate – [FSingleton StringType] This holds the time stamp when this Knowledge Base was openned – i.e. the start of this session.

NetworkStatus – [FSingleton NonNILType] This stores the last person to use this Knowledge Base, and when that last use was.

WhenOpeningNetwork – [FSingleton UnrestrictedType] This points to a list of functions which CORLL calls when openning this Knowledge Base. Each takes two arguments – the first is the name of this KB, and the second is passed from *WhenOpenningNetworks*, serves to suppress questions and messages.

LoadFiles – [FSet NonNILType] This lists the files which CORLL will read in whenever it opens this network. It will also ask if it should *MAKEFILE* these when this Knowledge Base is closed.

KBsFNS – [FSingleton NonNILType] This points to a variable whose value lists the functions relevant to this Knowledge Base.

WhenWritingNetwork – [FSingleton NonNILType] This points to a list of functions which CORLL calls when writing out this Knowledge Base. Each takes two arguments – the first is the name of this KB, and the second is passed from *WhenWritingNetworks* , serves to suppress questions and messages.

Networks – [FSet KBType] Each Knowledge Base may depend, hierarchically, on the presence of other knowledge bases, in core. The KBs <kb> requires are listed in <kb>.STATUS:*Networks* .

KBsUnitIndex – [FSingleton NonNILType] This points to the unit which holds the unit index CORLL uses for this Knowledge Base.

KBsFreeBlockIndex – [FSingleton NonNILType] This points to the unit which holds the free block index CORLL uses for this Knowledge Base.

DependentNetworks – [FSet KBType] This lists the Knowledge Bases which rely on the presence of this Knowledge Base to operate.

AnyProcess (*23*) - Every action which takes place, in LISP, is a *Process* . This corresponds to each function in LISP.

SuperClass: AnyCT&U

SubClass: AnyAction AnyFunction AnyUnitListProcess

The following slots are defined for all 'Process''s:

CVUsedBy – [FSet FunctionType] Having x in y:*CVUsedBy* means x s cached-value should be updated whenever y s cached-value changes.

Inverse: IUseCVOf

DefnUsedBy – [FSet FunctionType] Having x in y:*DefnUsedBy* means x s defn, and maybe its previously stored values, should be updated whenever y s defn changes

Inverse: IUseDefnOf

IUseCVOf – [NotAFormat NotARange] Having y in x:*IUseCVOf* means if the y stored value of some s should change, some x value may change as well.

Inverse: CVUsedBy

HighLevelDefn: (Apply*ingFn GetAllCVs HighLevelDefn)

IUseDefnOf – [NotAFormat NotARange] Having y in x:*IUseDefnOf* means x s defn, and maybe its previously stored values, should be updated whenever y s defn changes

Inverse: DefnUsedBy

HighLevelDefn: (Apply*ingFn GetAllFNS HighLevelDefn)

LispFn – [FSingleton FunctionType] The actual compiled code LISP will run, to process a process, is stored here.

HighLevelDefn: (OneOf LispFnForSlot LispFnForStoredFn)

HowToProcess – [FSingleton FunctionType] This will soon be deleted, in favor of LispFn.

WhatToProcess – [FSet UnrestrictedType] Fill in later.
HighLevelDefn: (OneOf TaskList RuleList)

VerifyArgs – [FSingleton FunctionType] The value of F:*VerifyArgs* is a function which is true if its argument is acceptable as input to the function represented by F .

AnyAction (24) - This includes any activity carried out in the real world by physical objects.
SuperClass: AnyProcess
SubClass: AnyActionSequence
Direct Examples: DescribeUnit

The following slots are defined for all "Action"s:

SpecAct – [FSet UnitType] This points to "refinements" of this action – i.e. activities which are more precisely specified.
Inverse: GenlAct

GenlAct – [FSet UnitType] This points to actions which are more general (i.e. at a higher level of abstraction) than the action encoded by this unit. E.g. Locomotions ∈Walking :*GenlAct* .
Inverse: SpecAct

*SpecAct** – [FSet UnitType] A list of a unit's SpecAct, THEIR SpecAct, etc.
Inverse: GenlAct*
HighLevelDefn: (Starring SpecAct)

*GenlAct** – [FSet UnitType] A list of a unit's GenlAct, THEIR GenlAct, etc.
Inverse: SpecAct*
HighLevelDefn: (Starring GenlAct)

AnyFunction (25) - Functions are distinquished from processes in that the primary purpose of a function is to return a value. Note a process may be run, in effect, for some side effect. (Yes, this is NOT pure LISP.)
SuperClass: AnyProcess
SubClass: AnyFunctional AnyPredicate AnyStorableFn AnyUnitListFn

The following slots are defined for all "Function"s:

SlotsUsedInBuilding – [FSet SlotType] Lists the slots which this one contributes to defining.
Inverse: AllSBF

HighLevelDefn – [FSingleton NonNILType] Here is stored a High Level Specification of the code to be run. This can be "parsed" into a piece of LISP code, which LISP can execute. Ideally, the information here should be sufficient to fully specify a function.

DataRange – [FSingleton NonNILType] The value of F:*DataRange* is used by the range interpreter associated with D , the *Datatype* of the function S , to generate a function capable of deciding whether a value is acceptable or not.

Datatype – [FSet (UnitType (*P AnyDatatype))] S:*Datatype* points to the list of units in the range of the function, F .
Inverse: IsTypeOf

Format – [FSingleton (UnitType (*P AnyFormat))] This stores the format of the result this function is expected to return.

FunctionCharacter – [FSet NonNILType] This holds facts which serve to describe this function. It is not currently used.

Defn – [FSingleton FunctionType] This function must take in a slot name s and return a function capable of reading/computing s in general. Ultimately, Defn:*Defn* should have a self-compiling call placed in each value it returns.
HighLevelDefn: (Apply*ingFn CAR FunctionSpec)

Definition – [FSingleton NonNILType] This is not currently used; and may be meaningless.

Domain – [FSet (UnitType (*P AnyClassOfObjects))] This points to units, each of which represent a class in the domain of this function.
Inverse: DomainOf

Range – [FSet (UnitType (*P AnyClassOfObjects))] This points to units, which each represent a class in the range of this function.
Inverse: RangeOf

IsBuiltFrom – [NotAFormat NotARange] Appears in unit X, for a type of slot, and lists the old things out of which X has been defined
HighLevelDefn: (Apply*ingFn AllButHead HighLevelDefn)

UnitsBuiltFrom – [NotAFormat NonNILType] This is going away soon.
Inverse: UsedInBuilding
HighLevelDefn: (Subsetting IsBuiltFrom Unitp)

UsingFunctionals – [NotAFormat NotARange] Appears as a slot in unit X, and tells how X was defined out of other slots
Inverse: CombinerFor
HighLevelDefn: (Subsetting UsingFunctions (MemberOf AllIsas AnyFunctional))

SlotsBuiltFrom – [NotAFormat NonNILType] Fill in later.
Inverse: SlotsUsedInBuilding
HighLevelDefn: (Subsetting UnitsBuiltFrom (MemberOf AllIsas AnySlot))

UsingFunctions – [NotAFormat NotARange] Appears as a slot in unit X, and tells how X was defined out of other slots
HighLevelDefn: (Apply*ingFn OnlyHead HighLevelDefn)

PreConditions – [FSet NonNILType] Fill in later.

DomainType – [FList NonNILType] This holds a type specification indicating the domain over which this function is defined.

RangeType – [FList NonNILType] This holds a type specificiation, indicating the range into which this function will map.

AnyFunctional (*26*) - Each descendant unit represents a function whose range is a space of functions.
SuperClass: AnyFunction
SubClass: AnyLogicalOp AnySlotCombiner
Direct Examples: ApplyToEach Apply*ingFn ApplyingFn MemberOf
The following slots are defined for all "Functional"s:

GetCVsUsed – [FSingleton FunctionType] The value of SC:*GetCVsUsed* is a function which, when applied to a high level defn, HLDefn, returns a list of storable functions on whose cached values this HLDefn depends. (SC is the *CAR* of that HLDefn.) This computed function can then be stored on the function S , which will usually be a slot.

GetFnsUsed – [FSingleton FunctionType] The value of SC:*GetFnsUsed* is a function which, when applied to a high level defn, HLDefn, returns a list of functions on whose definition this HLDefn depends. This can then be stored on the function S , which will usually be a slot.

CombinerFor – [FSet (UnitType (*P AnyFunction))] This slot appears in a unit X for a type of functions, and lists those slots which are defined out of old ones by using X
Inverse: UsingFunctionals

ToParseParts – [FSingleton FunctionType] This value is used by each Slot Combiner to parse its list of arguments.

AnyLogicalOp (*27*) - Fill in later.
SuperClass: AnyFunctional
Direct Examples: L-Optional L-NOT L-OR

AnySlotCombiner (*28*) - An operator which takes some old slots and defines a new one out of them.

SuperClass: AnySlotListFn AnyFunctional

Direct Examples: DoneIndirectly Listing PutInOrder Soften OneOf Plussing OrderedUnioning Unioning OrderedComposition OrderedStarring Composition Application Starring FirstOk Intersecting CommonXProd Subsetting

The following slots are defined for all "SlotCombiner"s:

FnForInverting – [FSingleton FunctionType] To find the inverse of a slot, one can examine the high level definition of that slot, and attempt to invert that. The *FnForInverting* slot of a slot combiner, SC , is a function which takes as an argument a high level definition, and returns the high level definition of a slot which computes the inverse functions from the original slot.

FnForUpdating – [FSingleton FunctionType] When a new value is placed in a slot, several other slots in the Knowledge Base must be updated. Such updates are performed by executing the code stored in the *KBUpates* slot of this slot. The *FnForUpdating* slot of a slot combiner is used to compute this *KBUpdates* . It takes a high level definition as its argument, with this particular slot combiner as principle slot combiner, and returns, essentially, the value for the *KBUpdates* slot for this slot (defined by that high level definition) .

FnForCaching – [FSingleton FunctionType] After the value of a slot has been computed (using the slot's definition,) RLL then considers storing that value away. Each slot combiner suggests an appropriate algorithm for deciding whether to store such values, and where. That procedure is encoded in the *FnForCaching* slot of the slot combiner. (This function takes a high level definition as an argument, and returns a function to fill the *ToCache* slot of a slot.)

AnyPredicate (*29*) - These functions return a value which is used as a Boolean – i.e. they serve to intensionally define a set.

SuperClass: AnyFunction

AnyStorableFn (*30*) - This class contains those functions whose value, on some input, might be stored, (or cached,) away. Note at least one of the arguments must be a unit.

SuperClass: AnyFunction

SubClass: AnyUnitListFn

The following slots are defined for all "StorableFn"s:

ToCache – [FSingleton FunctionType] The function stored here is called after a value has been calculated. This function then decides whether to store this value for future use, and if so, where.

StoredAList – [FSet (FSet UnrestrictedType)] This stores some i/o pairs for this function, as an association list.

LispFnForStoredFn – [FSingleton FunctionType] This function, used as the value for *LispFn* , does the following: First try to find the value by looking it up. If that fails, compute it; and consider caching the results. Of course, then return the computed (or retrieved) value.

ToInvalidate – [FSingleton FunctionType] The function stored here is used when some cached value is to be discarded.

ToConfirmValue – [FSingleton FunctionType] When retrieving a potential value for some input data, the predicate stored on this function's *ToConfirmValue* slot is used to see if this value is valid.

ToLookUp – [FSingleton FunctionType] The value of this slot is a function, which attempts to retrieve a cached value of this function.

AnyUnitListFn (*31*) - Here will be any mapping which takes, as an argument, one or more units

SuperClass: AnyStorableFn AnyFunction

SubClass: AnySlotGetter AnySlotListFn AnyUnitFunction

AnySlotGetter (*32*) - Examples are the units used to get the list of slot types which new units, created using some inheritance mechanism, should have.

SuperClass: AnyUnitListFn

Direct Examples: PossibleSlotsOfISubClass PossibleSlotsOfITypEx PossibleSlotsOfIExamples

AnySlotListFn (*33*) - Descendants are functions which takes one or more slots as arguments.

SuperClass: AnyUnitListFn

SubClass: AnySlotCombiner

AnyUnitFunction (*34*) - Descendants each represent a mapping which takes, as an argument, a unit.

SuperClass: AnyUnitListFn

SubClass: Any*SELF*Slot AnySlot

Direct Examples: FunctionSpec MyKB

The following slots are defined for all "UnitFunction"*s:*

HandDoneSBF – [FSet SlotType] This is used to enter the names of slots (or, in general, functions,) which current slot was built from.

UsingSlotCombiners – [NotAFormat NotARange] Appears as a slot in unit X, and tells how X was defined out of other slots

HighLevelDefn: (Subsetting UsingFunctionals (MemberOf AllIsas AnySlotCombiner))

AfterGetValue – [FSingleton FunctionType] The value of S:*AfterGetValue* is a function which is applied to a unit U the *S* and the value U:*S* , after determining this value. Any final test to be made, can be done here.

ToGetValue – [FSingleton FunctionType] The function stored on a function's *ToGetValue* is invoked when one requests x:*S* .

LispFnForSlot – [FSingleton FunctionType] The value of S:*LispFn* is a function, which, when applied to a unit, U , returns the value of U:*S* .

AllSBF – [FSet SlotType] This stands for All Slots Built From . It is used to hold the set of all slots which affect this one – that is, x:*S* may have to be invalidated if the *y* slot of some unit is changed, whenever y ∈S:*AllSBF* .

Inverse: SlotsUsedInBuilding

HighLevelDefn: (Unioning SlotsBuiltFrom HandDoneSBF)

BeforeGetValue – [FSingleton FunctionType] The value of S:*BeforeGetValue* is a function which is applied to a unit U and a *S* , and returns nonNIL if these arguments are appropriate.

ActualGetValue – [FSingleton FunctionType] The value of S:*ActualGetValue* is a function which is applied to a unit U and a *S* , and returns acutally does the retrieval of the vaue of U:*S* .

Any*SELF*Slot (*35*) - Descendants of this are the oft-spoken syntactic slots. That is, they each refer to this unit, *qua* unit, rather than what this unit represents. See AnyCT&U and AnyAT&U to understand this hackery.

SuperClass: AnyUnitFunction

There are currently 9 examples.

*The following slots are defined for all "SELF*Slot*"s:*

StoredInTypAs – [FSingleton SlotType] This points to slot which holds the inheritable value in typical example units.

Inverse: StandsForSlot

AnySlot (*36*) - Every function which takes a unit as an argument, and which MAY BE STORED ON THAT UNIT, is a slot; and descends from AnySlot .

SuperClass: AnyUnitFunction

SubClass: AnyComputableSlot AnyField AnyInheritableSlot AnyPrimitiveSlot

The following slots are defined for all 'Slot"s:

ComputeWhenFilled – [FSet SlotType] Whenever a new value fills U:*S* , the value of S:*T* should be recomputed for each T ∈S:*ComputeWhenFilled* .

IsEssentialFor – [FSet UnitType] Some virtual slots must be stored on a unit for bootstrapping reasons. S:*IsEssentialFor* holds a list of units which require this *S* slot.
Inverse: MyEssentialVirtualSlots

Inverse – [FSingleton SlotType] Stating S is the *Inverse* of a *T* means x ∈y:*S* iff y ∈x:*T* . The ∈ relation means a =b if b is a singleton, otherwise a is in the list, b.
Inverse: Inverse

SuperSlot – [FSet SlotType] Stating SS is a *SuperSlot* of *S* means the value of x:*S* will be a subset of the value of x:*SS* , for all x in their common range.
Inverse: SubSlot

SubSlot – [FSet UnitType] This is the inverse of *SuperSlot* .
Inverse: SuperSlot

MakesSenseFor – [FSet (UnitType (*P AnyArchetype))] A given slot, S , may only be defined for certain particular units. S:*MakesSenseFor* points to a list of typical-example units. This *S* slot makes sense for each instance of each such unit.
Inverse: NewPossibleSlots

ToPutValue – [FSingleton FunctionType] The function stored on S:*ToPutValue* is called whenever putting a new value onto U:*S* .

ToInitialize – [FSingleton FunctionType] When creating a new unit, all of the existing inheritance mechanisms first gather a collection of slots, which are meaningful to this new unit. Each slot, S , is then asked for its S:*ToInitialize* function, which is then run. It is the responsibility of this function to actually store an appropriate value on this new unit.

*SuperSlot** – [FSet UnitType] A list of a unit's SuperSlot, THEIR SuperSlot, etc.
Inverse: SubSlot*
HighLevelDefn: (Starring SuperSlot)

*SubSlot** – [FSet UnitType] A list of a unit's SubSlot, THEIR SubSlot, etc.
Inverse: SuperSlot*
HighLevelDefn: (Starring SubSlot)

ToAddValue – [FSingleton FunctionType] Whenever one wishes to add one a value to the current value of U:*S* , the function stored on S:*ToAddValue* is called.

ToDeleteValue – [FSingleton FunctionType] Whenever one wants to delete a value from the list of values stored on U:*S* , the function stored on S:*ToDeleteValue* is called.

ToSubstValue – [FSingleton FunctionType] Whenever one wants to substitute one value for another, on the list of values stored on U:*S* , the function stored on S:*ToSubstValue* is called.

KBUpdates – [FSingleton FunctionType] Whenever a value is stored in a slot, various changed must be made throughout the Knowledge Bases, for truth maintanence reasons. A function designed to perform such modifications is stored in the *KBUpdates* slot of each slot. S:*KBUpdates* is called whenever the value of x:*S* is changed.

This *KBUpdates* is calculated using the FnForUpdating slots of the various Slot Combiners used to define this S slot.

VerifyAll – [FSingleton FunctionType] Before accepting a value for storage on U:*S* , it is tested for acceptability. This is done by calling S:*VerifyAll* on this proposed value.

VerifyElement – [FSingleton FunctionType] When adding a new value to a slot's existing value, or substituting one value for another, it is often costly, and unnecessary, to check all of the values for acceptability. To verify the validity of one value, the function stored on S:*VerifyElement* is called on that proposed new element.

OrderForToInit – [FSingleton IntegerType] Each slot will have a value, stored here, which indicates at what time its ToInitialize function should be invoked when a new unit is being created. It may use the global variables: uParent, uInheritance, and uAllInheritantedSlots, to make its decision.

ActualPutValue – [FSingleton FunctionType] The value of S:*ActualPutValue* is a function which is applied to a unit U , a *S* , and a value V, actually stores V on U:*S* .

ActualAddValue – [FSingleton FunctionType] The value of S:*ActualAddValue* is a function which is applied to a unit U a *S* , and a value V, and returns actually does the addition of the value V to the U:*S* .

ActualDeleteValue – [FSingleton FunctionType] The value of S:*ActualDeleteValue* is a function which is applied to a unit U a *S* , and a value V, and returns actually does the deletion of the value V from the U:*S* .

ActualSubstValue – [FSingleton FunctionType] The value of S:*ActualSubstValue* is a function which is applied to a unit U a *S* , and values V and W, and returns actually does the substitution of the value W for the value V in the U:*S* .

AnyComputableSlot (*37*) - These slots are redundant, as they could have been computed from other, more basic slots. (Modulo Garden-Of-Eden conditions. See MyEssentialVirtualSlots .)

SuperClass: AnySlot

There are currently 72 examples.

AnyField (*38*) - Slots on sub-units (that is, units devoted to storing the value of a slot of a given unit,) are called fields. Those "slots" which appear only in this context are stored under AnyField .

SuperClass: AnySlot

Direct Examples: *vaLue* LivesInLocation LivesInSlot LivesInUnit

AnyInheritableSlot (39) - Descendants of this class are slots whose value may be inherited from some prototype of the unit in question. Or course, if there is a value stored on that unit, that value will be used.

SuperClass: AnySlot

SubClass: AnyAccessSlot AnyFormatFnSlot

Direct Examples: ToLookUp ToConfirmValue OrderForToInit

AnyAccessSlot (40) - Descendant of this AnyAccessSlot are slots used to manipulate the units themselves. For efficiency, they all use the same fast retrieval mechanism to determine their respective values – *GetAccessFn* .

SuperClass: AnyInheritableSlot

Direct Examples: ToInvalidate ActualSubstValue ActualDeleteValue ActualAddValue ActualGetValue ActualPutValue ToGetValue BeforeGetValue AfterGetValue ToPutValue BeforePutValue AfterPutValue ToAddValue ToDeleteValue ToSubstValue ToInitialize MyToKillMe ToCacheField ToCache ToKillValue MyToRenameMe

AnyFormatFnSlot (41) - Various bits of information are associated with each format. When this information is functional, it descends from this AnyFormatFnSlot .

SuperClass: AnyInheritableSlot

There are currently 8 examples.

AnyPrimitiveSlot (42) - Primitive slots, which descend from this AnyPrimitiveSlot , cannot be computed if omitted. (As opposed to computable slots, which are technically redundant information, as they can be computed from more basic slots.)

SuperClass: AnySlot

SubClass: AnySlot-Instances

There are currently 96 examples.

AnySlot-Instances (43) - Aliases used for syntactic slots are stored under AnySlot-Instances . These are used to hold values which should be inherited from typical example units; freeing the basic slot to hold the value pertanent to this particular unit.

SuperClass: AnyPrimitiveSlot

Direct Examples: SlotsNowOrdered-Instances EssentialVirtualSlots-Instances ToKillMe-Instances ToRenameMe-Instances

The following slots are defined for all "Slot-Instances"s:

StandsForSlot – [FSingleton SlotType] This points to the name of the syntactic slot for which this is an alias.

Inverse: StoredInTypAs

AnyUnitListProcess (44) - These processes take one or more units (amoung possibly other things) are arguments.
SuperClass: AnyProcess
SubClass: AnyUnitProcess

AnyUnitProcess (45) - These processes take a single unit as its argument.
SuperClass: AnyUnitListProcess
Direct Examples: EditUnit

AnyUnit (46) - Examples will be things which REPRESENT units... NOTE: this does NOT include every unit automatically! (In fact, most units represent some real world object, such as Tree#32, or some conceptual entity, such as Red, or Function#412.)
SuperClass: Anything

The following slots are defined for all "Unit"s:

MyCreatedAs - [FListN (UnitType (*P AnyInheritance)) (FList UnitType)] This stores inheritance information about this unit - indicating, for example, that it was created as an IExamples (read "Example") of AnySlot .

MyEssentialVirtualSlots - [FSet SlotType] These slots are essential for the Garden of Eden RLL system. Therefore *RemoveVirtualSlots* is smart enough to know NOT to remove these slots (i.e. those which *MyEssentialVirtualSlots* points to) from a unit.
Inverse: IsEssentialFor

MyTimeOfCreation - [FSingleton StringType] This records when this unit was created.

MyCreator - [FSingleton StringType] This names the user who created this unit.

MyToKillMe - [FSingleton FunctionType] This function is called when deleting this unit.

MySlotsNowOrdered - [FSet SlotType] This lists the names of slots which are currently in the correct order. (For example, *OrderedPrototypes* appears in some unit's *MyEssentialVirtualSlots* only when the typical example units stored in *Prototypes* have been arranged in the correct order.)

MySensibleSlots - [FSet SlotType] Only certain slots are defined for a given unit. This list is stored in that unit's *MySensibleSlots* .
HighLevelDefn: (Composition NewPossibleSlots Prototypes)

MySlots - [FSet SlotType] This never cached slot returns the list of slots belonging to this unit.
Inverse: AmUsedIn

MyToRenameMe - [FSingleton FunctionType] This function is called when renaming this unit to another name.

AnyUser (47) - RLL tries to hold some primitive information about each user of this system. A unit is devoted to each user, (as well as each recognised user class) ; and this information is under AnyUser .

SuperClass: AnyCT&U

SubClass: AnyHacker

Direct Examples: AndyFreeman LarryHines

The following slots are defined for all "User"s:

InformalName – [FSingleton StringType] This is a name RLL can use to greet this user.

UsualKBs – [FSet KBType] These are the Knowledge Bases this user usually wants loaded in.

WritingOptions – [FList NonNILType] When closing a Knowledge Base, RLL must ask the user several questions. To sidestep this tedious (and often unneccessary) process, the user can indicate a fixed set of responses to such inquires; which are stored on this *WritingOptions* slot. When closing the KBs, the user is now asked a single question – if he wishes to use these. (Answering No forces RLL to ask him these questions one by one.)

The defaulted writing function, *StandardFinishUp* , asks if virtual slots should be removed, if this KB should be disconnected from the others, and if this KB should be diagnosed; in that order. Setting *WritingOptions* to (Y N Y) instructs RLL to remove virtual slots, and diagnose the KB, but not to disconnect it.

OpenningOptions – [FList UnrestrictedType] Like *WritingOptions* , this helps the user to avoid a potentially dull task. The value stored here will be handed to the function called when openning each knowledge base; if the user indicates he wishes his default setting to be used.

The only question *StandardStartUp* might ask is whethr to reconnect an enterring knowledge base. Setting the *OpenningOptions* slot to (NIL) means this question will be asked each time.

UserNames – [FSet StringType] This lists the system names this user may go by. (Ie values of *(USERNAME)* which correspond to this person.)

AnyHacker (48) - This class includes people working on RLL.

SuperClass: AnyUser

Direct Examples: DougLenat RussGreiner

AnyConcreteThing (49) - Instances refer to tangible objects, (such as trees,) as opposed to abstract things (such as variables) .

SuperClass: Anything

E.4: INDEX of UNITS.

For indexing purposes, the classes shown in Section E.3 were numbered sequentially. This value is used in the index below, to indicate in which class each of these units (representing classes, their examples and significant slots) belong.

F. APPENDIX - Environment

The functions most RLL-1 users will need fit into three basic groups. Those which a novice user should know are listed first, organized by topic. These top level functions place essentially no restriction on the nature of the knowlede base on which they are used. The next group of functions are one level deeper, consisting of the functions required for bootstrapping RLL-1. These are listed alphabetically. Most of these functions live in some unit, and are used by default -- i.e. unless overwritten. The final category are utility functions, which augment LISP in useful ways.

The rest of this appendix list miscellaneous functions which have been adviced or altered and relevant global variables. The various functions and variables which comprise CORLL, (see [*Smith*]) may be used as well. (Recall RLL-1 is built on this unit-management system.)

These functions will, in general, return NIL only when some error has been encountered -- for example, when the slot in question is not really a *bona fide* slot. Also, many of the parameters mentioned below are optional, and serve only to speed up the functions processing, if supplied. To indicate this distinction, each required parameter will begin with a capital letter, while each extra one will start with a lower case letter.

Thanks to a special "hack" made to LISP's interpreter, many of the units can serve as functions. Seeing (FN arg_1 arg_2 ... arg_N), LISP will first attempt to apply the functional definition of FN (i.e. the lambda expression stored in (GETD 'FN)) to these arguments. If (GETD 'FN) is NIL, before raising an error interrupt, LISP will then check if FN is a Process -- that is, a unit which descends from *AnyProcess*. If so, and if the arguments, (arg_1 arg_2 ... arg_N), are in the domain of FN, (iff (APPLY* (GetValue 'FN 'VerifyArgs) arg_1 arg_2 ... arg_N) is nonNIL), LISP will (APPLY* (GetValue 'FN 'LispFn) arg_1 arg_2 ... arg_N), and return that result.

As mentioned in subsection 5.5, much of the "smarts" of RLL-1 has been relegated to some unit, as opposed to the more standard practice of simply coding it opaguely into some (set of) functions. For example, there is no mention below of Fields, (or any other indirect pointers,) which Appendix B.4 implied must exist. This information has been placed within the FSeeUnit unit; which "knows" how to access such values, and how to modify them. Hence we saw the value physically stored in the *Age* slot of *Mary* was "("Do" FSeeUnits 39 (U4S AgeOfMary0001) (U4S AgeOfMary002))". (The initial "Do" indicates this is a special slot value; see Appendix D.4.) This particular mechanism, of using "value formats", has other, less epistemologically motivated uses. For example, to indicate the *Color of George* is either Green or Red, on can put the value "("Do" FCneOf Gree Red)" into George's *Color* slot.

D.1: Top Level Functions

Overhead:

Unitp[Unit] -- Returns nonNIL iff Unit is a unit, belonging to one of the networks currently loaded in.

Processp[Process] -- Returns nonNIL iff *Process* is a process which belong to one of the networks currently loaded in. As this is often involved in the *VerifyArgs* test described above, any Knowledge Base which uses processes is expected to provide RLL-1 with such a function. (This happens to reside in ProcessType:*VerifyType*.)

Slotp[Slot] -- Like Processp, this returns nonNIL iff *Slot* is a slot which belong to one of the networks currently loaded in. Again, any Knowledge Base which uses slots is expected to provide RLL-1 with such a function. (This happens to reside in SlotType:*VerifyType*.)

EDITU[Unit extra] - Calls the LISP editor on the slots and their values of this unit. They will be arranged as a property list. The effect will be the same as actually performing a PutValue[Unit Slot Value old extra] for each slot changed, where Value is the new value stored in this slot, and old is either the value which had been there or RecomputeMe.

Notes: EDITU is an NLAMBDA, *ala* EDITP.
Like EP, "EU" is a LISPXMACRO, as well as a USERMACRO.
One can terminate the editting session, without performing the changes, by typing "ABORT".
Typing "P-A *x y z*" will reset extra to the value *(x y z)*.
(P-A) resets extra to NIL.
Ending a session with "SimplePut" will use the CORLL function UA-PUTVALUE, rather than the full PutValue.

DI[Unit other depth] - Prints out the (psuedo-) hierarchy of units, starting with the unit, Unit. (If omitted, will start from the root, called Anything.) The optional list, other, specifies which branches to take on the descent. As with Slotp, each Knowledge Base should supply this function. (In this implementation,) if other includes any of {S SubClass Specs}, all of the Specializations (which is usually SubClass) of each unit walked will be examined. Any of {T Typ TypicalExample} will cause DI to print out the typical example of each class of each unit walked. {E Examples} both print out of the UnitExamples of each class unit walked. (By default DI will follow all of these links.) This will stop recur(s)ing down the tree after (CR depth 1000) iterations.

START[ ] - Starts the RLL-1 system; this will load in the desired Knowledge Bases, and perform other initializing functions.

GetKBs[ ] -- This function loads in the RLL Knowledge Base, and then the others the user has requested.

Close[ ] -- Closes each Knowledge Base now open.

CANCEL[ ] -- Cancels each Knowledge Base now open.

CC[ ] -- Asks the user whether it should Close or Cancel each Knowledge Base now open, and does so.

Retrieval:

Slot's value

GetValue[Unit Slot others] -- This is RLL-1's basic get. It returns the value derived by applying Slot:*ToGetValue* to Unit, Slot and others. (The semantics of the optional third argument depends on that stored function.) As this slot is accessible to the user, he can code arbitrarily complex retreival schemes. The current value of *ToGetValue*:*ToGetValue* is GetAccessFn. As shown above, (in Appendix A.1, or subsection 4.3) the result of the function call, (GetAccessFn Slot *ToGetValue*), is applied to (Unit Slot others). The value this returns is returned by the overall GetValue.

Writing:

Slot's value

PutValue[Unit Slot Value oldvalue why] -- This is RLL-1's vanilla put function. It calls the function Slot:*ToPutValue*, handing it all the arguments listed above. [Recall that only the first three

arguments are required; of the others, (those beginning with a lower case letter,) oldvalue will be computed if it is not given and found to be necessary.]

AddValue[Unit Slot Value oldvalue why extra] -- This function is used to add a new value to the list of entries already stored as Unit:*Slot*, using the function stored on Slot:*ToAddValue*. By default, (i.e. stored on TypicalSlot:*ToAddValue*), DefaultAddValue will be called on this argument list. The variable "why" holds information describing why this operation was performed; and "extra" is used to additional (non-why) data to the actual adding function. (Note DeleteValue and SubstValue use these same two extra arguments, for the same purpose.)

DeleteValue[Unit Slot Value oldvalue why extra] -- Like AddValue, this function is designed to remove a value from the list stored on Unit:*Slot*, using Slot:*ToDeleteValue*. By default, (i.e. stored on TypicalSlot:*ToDeleteValue*), DefaultDeleteValue will be called on this argument list.

KillValue[Unit Slot oldvalue why] -- This removes all traces of the slot Slot from the unit Unit. It should be the same as performing a PutValue[Unit Slot RecomputeMe oldvalue context why]. It actually uses the Slot:*ToKillValue*, which defaults to the value stored on TypicalSlot:*ToKillValue*, DefaultKillValue.

SubstValue[Unit Slot ToValue FromValue oldvalue why extra] -- This substitutes the value FromValue with the value ToValue in Unit:*Slot*. The value of Slot:*ToSubstValue*, which defaults to DefaultSubstValue, may be used.

CacheValue[Unit Slot Value why] -- This is the command issued which considers storing Value on Unit:Slot. By default, (i.e. stored on TypicalSlot:*ToCacheValue*), DefaultSlotCacher will be called on this argument list.

KB Management:

CreateUnit[Unit KnowledgeBase] -- This creates a new unit, named Unit, adding it to the Knowledge Base, KnowledgeBase.

NewUnit[Unit Inheritance ParentSet KnowledgeBase] -- This creates a new unit, named Unit, which is a descendent of each member of the ParentSet, by the inheritance, Inheritance. After some preliminary overhead, it calls InitializeUnit, whose mechanism has yet to be "officially" decided..

KillUnit[Unit] -- Deletes the unit named Unit, disconnecting all of its links. (The end result should be as if this unit had never existed.) It really calls Unit:*MyToKillMe*, which defaults (when TypicalUnit is reached) to DefaultKillUnit.

RenameUnit[NewName Unit KB] -- This changes the name of the unit, Unit, to be NewName, in the Knowledge Base, KB, and propagates the effects of this change. It really calls Unit:*MyToRenameMe*, which defaults (when TypicalUnit is reached) to DefaultRenameUnit. (This will often be intercepted, by, for example, the value of TypicalCLass:MyToRenameMe.)

D.2: Functions Needed to Bootstrap RLL-1

Many of these functions reside somewhere in the RLL-1 *Knowledge Base. This information is listed below; following the list of arguments.*

AddInverseLinks[Unit] -- This add the inverse links (back pointers) to every link emanating from Unit.

CacheIfOK[Unit Slot Value oldValue why] -- Caches Value in Unit:*Slot* only if IsOk[Value].

CacheIfNonTrivial[Unit Slot Value oldValue why] -- Caches Value in Unit:*Slot* only if IsOk[Value] and Value does not equal (LIST Unit).

CleanWS[ ] -- This tells CORLL that no unit has been written out. The first call to the CORLL function UP-PUTUNIT will "dirty" the current work space; undoing this fact.

CreateSlot[HLDefn KB sv] -- This creates a new slot, whose high level definition is HLDefn. This new slot will be stored in the knowledge base, KB. The user may optionally add other slots to this new unit; these would be passed as a list of dotted pairs, in the sv argument.

DefaultActualAddValue [Unit Slot Value oldvalue why extra] -- {TypicalSlot:*ActualAddValue*} This function actually adds on a new value, Value, onto Unit:Slot, whose value is now oldvalue. It will call on the appropriate slot format (or value format) as required.

DefaultActualDeleteValue[Unit Slot Value oldvalue why] -- {TypicalSlot:*ActualDeleteValue*} This actually deletes an old value. It is, in form, essentially identical to DefaultActualAddValue, guided once again by Slot:*Format*.

DefaultActualGetValue[Unit Slot others] -- {TypicalSlot:*ActualGetValue*}
This is used to actually retrieve (and recompute, if necessary,) the value of the Slot slot of the unit, Unit. The others argument contains a list of zero or more values, which are used modify this process; for efficiency reasons. The next two paragraphs will describe what DefaultActualGetvalue will do when others is NIL.

It will first use UA-GETVALUE to find if any value is physically stored on the *Slot* slot of Unit. If that value satisfies MustRecomputep, (i.e. is NIL, or RecomputeMe,) DefaultActualGetValue will apply Slot:*Defn* to Unit and Slot, and locally store the result. The function CacheValue is then called on Unit, Slot and this stored value. This function, described below, may physically cache this value in the knowledge base. In any event, that value is the actual value of Unit:*Slot*, and will be used in what follows.

If Unit:*Slot* is a special slot value, (i.e. of the form ("Do" <value-format> ...),) the value of the *FnForGetting* slot of <value-format> will be called on this value, as well as Unit, Slot and the *FnForGetting* slot of Slot:*Format*. Otherwise these arguments will be handed to that *FnForGetting* slot of Slot:*Format*. The value this call returns will be the result of this call.

Now for the exceptions: When FAST-GET is included in others, no *Defn*s will be tested -- and hence no writes will be performed into the Knowledge Base. Including NO-CACHE means no computed values will be cached (i.e. CacheValue will not be called), although such values will be computed as necessary. "IGNCRE-CACHE" tells DefaultActualGetValue to ignore any value stored in the *Slot* slot, and rely on Slot:*Defn*. If FAST-CACHE is included, it will be passed, as Fast-Cache, to that caching function. When IMPURE is in that list, the global variables uValue and uContext will be set to the value here returned, and the unit on which this value was physically stored, respectively. The remaining values are all used to speed up runtime execution. NO-VALUE-FORMAT indicates to return the value found, rather than call the Value Format's *FnForGetting*, (when that would have been applicable).

DefaultActualPutValue[Unit Slot Value oldvalue why] -- {TypicalSlot:*ActualPutValue*}
This function performs the actual put. The specifics of how this is done depends on whether Value or oldvalue has a value format, and on F, the *Format* slot of Slot. If both values are unformatted, the *FnForPutting* associated with the unit F is called on appropriate arguments. If either satisfies ValueFormattedp, the *FnForPutting* slot of that value format will be

called on these arguments, augmented with F:*FnForPutting*.

DefaultActualSubstValue[Unit Slot Value Modification why] -- {TypicalSlot:*ActualSubstValue*}
This actually performs the substitute of a new value for an old one, within the existing value of a Unit:*Slot*. The exact substituting process depends on the nature of Slot (in particular, on the value of Slot:*Format*).

DefaultAddValue [Unit Slot Value oldvalue why extra] -- {TypicalSlot:*ToAddValue*}
This is the default way to add a new value. Like DefaultPutValue, it will call DefaultBeforePutValue before performing this addition. (Here the value of single will be nonNIL, and is used to indicate the nature of this alteration.) The exact adding process is performed by DefaultActualAddValue. If this succeeds, DefaultAfterPutValue is called with these arguments.
"extra" tells whether a single value, or a list of values, are being added; and whether the current value of Unit:*Slot* should be recomputed (using Slot:*Defn*) if it is currently empty.

DefaultAfterGetValue [Unit Slot others] -- {TypicalSlot:*AfterGetValue*}
This is the default function executed after the actual getting within DefaultGetValue is performed. (DefaultGetValue will return NIL if this function does.) Currently this function simply returns T.

DefaultAfterPutValue [Unit Slot NewValue modif why] -- {TypicalSlot:*AfterPutValue*}
This is the default function executed after the actual putting within DefaultPutValue is performed. (DefaultPutValue will return NIL if this function does.) This function is responsible for maintaining KB consistency, using UpdateInverse and UpdateDepend as appropriate. It will also recompute the value of any essential virtual slot which was just now deleted; and will determine the value of certain other slots in a when-filled manner.

DefaultBeforeGetValue [Unit Slot others] -- {TypicalSlot:*BeforeGetValue*}
This is the default function executed before the actual getting within DefaultGetValue is performed. (DefaultGetValue will return NIL if this function does.) Currently this function simply returns T.

DefaultBeforePutValue [Unit Slot OldValue modif why] -- {TypicalSlot:*BeforePutValue*}
This is the default function executed before the actual putting within a DefaultPutValue is performed. (DefaultPutValue will return NIL if this function does.) If this value is being enterred by the user, this function will see if OldValue is an acceptable value, using (by default) DefaultVerifyValue.

DefaultDeleteClass [Unit] -- {TypicalClass:*MyToKillMe*}
This is the function used for deleting a unit which represents a class. In addition to deleting inverse links, this attempts to reclassify every example of this set, if the user gives the go ahead.

DefaultDeleteSlot [Unit] -- {TypicalSlot:*MyToKillMe*}
This is the function used for deleting a unit which represents a slot. In addition to deleting inverse links, this attempts to remove every occurance of this slot, if the user gives the go ahead.

DefaultDeleteU4S [Unit] -- {TypicalUnitForSlot:*MyToKillMe*}
This is the function used for deleting a unit which represents a slot's value. In addition to deleting inverse links, this automatically resets the value of the slot in the host unit.

DefaultDeleteUnit [Unit] -- {TypicalThing:*MyToKillMe*}
This is the default function used for deleting units in general. All it does is delete inverse

links.

DefaultGetValue[Unit Slot others] -- {TypicalSlot:*ToGetValue*}
This is used to retrieve (and recompute, if necessary,) the value of the Slot slot of the unit, Unit. The others argument contains a list of zero or more values, which are used modify this process; for efficiency reasons. The next paragraph will describe what DefaultGetvalue will do when others is NIL.

After performing an initial type check on its first two arguments, DefaultGetValue will apply Slot:*BeforeGetValue* to these arguments. If that returned nonNIL, the function stored on Slot:*ActualGetValue* is called. The value this call returns will be saved. Finally, Slot:*AfterGetValue* is called on this value, Unit and Slot. If all of these subfunctions succeed (i.e. returned nonNIL), that stored value will be returned.

Now for the exceptions: When FAST-GET is included in others, the functions on Slot:*BeforeGetValue* and Slot:*AfterGetValue* will not be executed. The remaining values are all used to speed up runtime execution. SAFESLOT and SAFEUNIT are designed to perform the MustBeSlot and MustBeUnit checks, on Unit and Slot, respectively, at compile time. They are only applicable if that parameter is a constant, i.e. is QUOTEd. SAFE is the same as including both of these. VERYSAFESLOT and VERYSAFEUNIT totally avoid calling MustBe--- functions, even at compile time. CONSTANTSLOT means we may assume the values stored on the unit Slot will not change from now on. CONSTANTUNIT makes similiar assumptions about the unit, Unit. These can be undone using CAUTIOUS, which insists on checking the validity of the arguments.

DefaultDeleteValue [Unit Slot Value oldvalue why] -- {TypicalSlot:*ToDeleteValue*}
This is the default way to delete an old value. It is, in form, essentially identical to DefaultAddValue.

DefaultKillValue [Unit Slot Value context why] -- {TypicalSlot:*ToKillValue*}
This is the default way to kill a slot and its value. It is, in form, essentially identical to DefaultPutValue, only using the *FnForKillingValue* slot of the *Format* slot of Slot, rather than its *FnForPutting* slot.

DefaultPutValue[Unit Slot Value oldvalue why] -- {TypicalSlot:*ToPutValue*}
This function is used for putting a value in a slot. Usually, it first calls Slot:*BeforePutValue* on the above argument list, augmented with one additional argument, described below. In an errorfree run, this will succeed (i.e. will return nonNIL,) and the actual writing will be performed by the function stored on Slot:*ActualPutValue*. Slot:*AfterPutValue* is then called on the same argument list as Slot:*BeforePutValue*, with Value substituted for oldValue.

If why incluces Fast-Put, the before and after functions will not be called. The extra argument expected by these functions (Slot:*BeforePutValue* and Slot:*AfterPutValue*) appears before why, "single". Its value should be NIL in this case, and the other parameters should be their values when handed to PutValue. [single's use will be apparent from AddValue, DeleteValue, and SuostValue, defined in this section.]

DefaultRenameClass[NewName OldClass] -- {TypicalClass:*MyToRenameMe*}
This is used to rename a unit which represents a class to a new name, and then performing the necessary KB updates. It will, if permitted, scan through each unit loaded it, attempting to perform this substitution. Otherwise, it will simply follow its various pointers, and change their value of their respective back-pointers to reflect this change. It will then ask if it should update first each function, and then all variables in the system; very time consuming processes.

DefaultRenameSlot[NewName OldSlot] -- {TypicalSlot:*MyToRenameMe*}
This is used to rename a unit which represents a slot to a new name, and then performing the necessary KB updates. It will, if permitted, scan through each unit loaded it, attempting to perform this substitution. Otherwise, it will do two things: First, like DefaultRenameUnit, it will follow its various pointers, and change their value of their respecuve back-pointers to

reflect this change. Second, it wil, unless prohibited, go to each unit which has this slot, and rename that slot. It will then ask if it should update first each function, and then all variables in the system; very time consuming processes.

DefaultRenameUnit[NewName OldUnit] -- {TypicalUnit:*MyToRenameMe*}
This is used to rename a unit which represents a class to a new name, and then performing the necessary KB updates. It will, if permitted, scan through each unit loaded it, attempting to perform this substitution. Otherwise, it will simply follow its various pointers, and change their value of their respective back-pointers to reflect this change. It will then ask if it should update first each function, and then all variables in the system; a very time consuming processes.

DefaultSlotCacher[Unit Slot Value Modification Why] -- {TypicalSlot:*ToCache*}
This simply stores Value on Unit:*Slot*, using UA-PUTVALUE.

DefaultSubstValue[Unit Slot Value Modification why] -- {TypicalSlot:*ToSubstValue*}
This is the default way to substitute a new value for an old one, within the existing value of a Unit:*Slot*. Like DefaultPutValue, it will call DefaultBeforePutValue before performing this addition. (Here the value of single will be nonNIL, and is used to indicate the nature of this alteration.) Next Slot:*ActualSubstValue* will be called on these arguments; and if this succeeds, DefaultAfterPutValue is called with these arguments.
"extra" tells whether a single value, or a list of values, are being changed; and whether the current value of Unit:*Slot* should be recomputed (using Slot:*Defn*) if it is currently empty

DefaultVerifyValue[Unit Slot OldValue Modification why] -- This function is used to verify that Modification represents a legitimate modification to the value, OldValue, now on Unit:*Slot*. It basically calls the function stored on the *VerifyAll* slot of Slot on the new value, if this change is replacing the full entry, or uses Slot *VerifyElement*, if we are simply adding a new value. If either the new value, or OldValue, was a value format, then it calls the *FnForVerifyingAll* (respectively *FnForVerifyingElement*) slot of the unit encoding that value format, on these argument, including that just computed Slot:*VerifyAll* (respectively Slot:*VerifyElement*).

FindDefault[Unit Slot other] -- This ascends Unit's *OrderedPrototypes*, asking each such typical unit for its Slot value. The first one it finds is returned.

FindInverse[HLDefn KB sv] -- This attempts to find a slot which is the inverse of the slot defined by the high level definition, HLDefn. If no such slot exists, and makes sense (i.e. HLDefn is invertable,) it asks if it should create such as slot. If the user permits this, it calls CreateSlot on that inverse, KB and sv. Otherwise it returns that high level definition.

FindSlot[HLDefn KB sv] -- This attempts to find a slot which is defined by the high level definition, HLDefn. If found, it returns the name of that slot.

FindValue[Value Unit Slot others] -- This auxillary function is used to determine the real value of Unit:*Slot* from the value physically stored there, assumed to be the first argument, Value.

FormattedValuep[Value] -- This returns nonNIL if Value is a value format -- i.e. is of the form ("Do" ...).

GetAccessFn[Unit Slot other default] -- Used to get various slots associated with accessing/updates values. Returns the first of {Unit *Slot*, FindDefault[Unit Slot other], default}, which passes IsOk[val].

HLDefnParser[ParseStr] -- Returns the functional specification derived from the High Level Definition, ParseStr. This is used by FunctionSpec *Defn*.

HLTypeParser[ParseStr Num Argname] -- This returns the body of a function which is designed to take an argument, and return nonNIL if that argument is of the type defined by ParseStr. "Argname" is the name of that argument, which that function should use. "Num" is either *FnForVerifyingAll*, or *FnForVerifyingElement*, depending on whether this eventual function should take a single element of the type, or the full value.

IsEmpty[Value] -- Returns nonNIL if value is one of RLL-1's substitute values used for NIL -- e.g. NoEntry or NoEntries.

IsOk[Value] -- Return NIL if value is either empty or needs to be recomputed. Otherwise value is returned.

InitializeUnit[Unit Inheritance ParentSet] -- This creates a new unit, named Unit, which is a descendent of each member of the ParentSet, by the inheritance, Inheritance. Its algorithm: Apply Inheritance:*GetPossibleSlots* to ParentSet. This produces a list whose elements are of the form (slotname location $where_1$ ... $where_N$). "location" is the name of the unit whose *ToInitialize* slot should be called, on Unit and slotname, to initialize that slot's value. Each $where_i$ is the name of a prototype of this new Unit, in which the slot, slotname, is first defined. InitializeUnit then maps along this list, calling InitializeSlot on slotname, ParentSet, location, and the list ($where_1$ ... $where_N$).

InitializeSlot[Unit Slot Location Protos] -- This invokes Location:*ToInitialize* on Unit, Slot and Protos (Protos is that Where-List). This may use the global variables: gParentSet, which is the parents of this unit, and gInheritance, which is the type of inheritance used.

IntersectDT[DTlist] -- Each element of DTlist is a datatype specification; this function returns a new type specification, which is the intersection of all of these -- i.e. whose implied acceptance criteria is the AND-junction of each member of the input list of datatypes.

InvertHLD[HLDefn] -- This returns the high level defition which computes the inverse function from the one implied by HLDefn, or NIL if such a function is undefined.

MapSlots[Fn] -- Maps along the slots of Unit, applying the function, Fn, to each slot and its value.

MapUnits[Fn kbs] -- Maps along all of the unit in any of the Knowledge Bases listed in kbs, applying the function, Fn, to each. If kbs is NIL, applies Fn to every resident unit. A companion function, MapUnitsQ quotes its arguments. If the first argument is not a LAMBDA expression, the second argument should be the name of the variable which is bound to the name of the current unit. This mapping extends to every resident unit.

MustComputep[value] -- Returns nonNIL if value indicates it is not current and should be recomputed, i.e. if value equals RecomputeMe or NIL. (value is usually some UnitSlot entry.)

NewIsa[NewUnit ParentUnits KB] -- This creates a new unit, NewUnit, as an instance of each of ParentUnits. It is entered into the Knowledge Base KB.

NewKB[NewKB] -- This creates and initializes a new Knowledge Base NewKB.

NewSubClass[NewUnit ParentUnits KB] -- This creates a new unit, NewUnit, as a subclass of each of ParentUnits. It is entered into the Knowledge Base KB.

NewTypEx[NewUnit ParentUnit KB] -- This creates a new unit, NewUnit, as a typical example of ParentUnit. It is entered into the Knowledge Base KB.

NotRemovable[Unit Slot Value ? ?] -- This returns nonNIL if the value of Unit.Slot must NOT be removed -- i.e. if it is essential.

NU[NewUnit FromUnit KB] -- This creates a new unit, NewUnit, essentially identical to FromUnit (after substituting NewUnit for each occurance of FromUnit). It is entered into the Knowledge Base KB.

OverallStartUp[ ] -- This asks the user if he wishes to start the system; and does so if allowed. It also does various other clean-up jobs, appropriate for restarting a suspended version of the RLL-1 system (i.e. after a SYSOUT)..

RemoveVirtualSlots[UnitList] -- This walks along the Unit in UnitList, removing their extraneous virtual slots. (Note a slot is essential if it is included in the unit's *MyEssentialVirtualSlots* list, or under a few other hacky conditions. See the function, NotRemovable.) There are several situations in which the user is prompted -- such as when an undefined slot encountered, or when this unit lacked some essential slot.

StandardFinishUp[KB args] -- {TypicalStatus: *WhenWritingNetwork*}
This function is called when the knowledge base, KB, is written out, and closed. It first asks if extraneous virtual slots should be removed, then, (if there are other currently open knowledge bases,) if KB should be disconnected, and finally, if this knowledge base should be diagnosed. These inquiries can be avoided if (CAR (NTH args i)) is Y (indicating an affirmative response) or N (negative).

StandardStartUp[KB args] -- {TypicalStatus: *WhenOpenningNetwork*}
This function is called when the knowledge base, KB, is openned. It first does various overhead tasks, such as initializing and updating variables. It then considers adding inverse links to all the units of KB, to reconnect to the other knowledge bases currently in core. This time consuming process is performed if there is some active knowledge base currently unconnected to KB, and the user permits it. This inquiry can be avoided if (CAR args) = Y (to connect the links if necessary) or N (not to reconnect, period).

START[] -- This actually begins RLL-1. It handles all of the overhead, then loads in the desired knowledge bases.

UnionDT[DTlist] --Each element of DTlist is a datatype specification: this function returns a new type specification, which is the union of all of these -- i.e. whose implied acceptance criteria is the OR-junction of each member of the input list of datatypes.

UpdateDepend[Unit Slot NewValue Modification why] --This is used to update the knowledge base, to reflect the Modification made to Unit:*Slot*. If appropriate, it calls the function stored on Slot:*KBUpdates*, on the above quintet of arguments.

UpdateInverse[Unit Slot NewValue Modification why addV delV] -- This is used to update the knowledge base, to reflect the Modification made to Unit:*Slot*. If Slot has an inverse, the back pointers are changed, as appropriate.

D.3: Convenience Functions

Advisor[ ] -- This advises several functions, for several reasons. See Section D.5.

EveryFn[List Fn] -- This is like MAPCAR, except it stops if applying Fn to any element returns NIL.

IntersectN[$List_1$ $List_2$ ... $List_N$] -- This is the *N*-ary form of the binary INTERSECT.

MapInsert[List Fn] -- This is like SubSetFn, except it breaks (i.e. gives a warning message) if any element of Fn fails IsOk, and it uses less LIST-space.

MapMerge[List Fn] -- This MERGEs the results of applying Fn to each element of List. (Note it breaks (i.e. gives a warning message) if any such element fails IsOk.)

MapUnion[List Fn] -- This UNIONs the results of applying Fn to each element of List. (Note it breaks (i.e. gives a warning message) if any such element fails IsOk.)

MapUntilOk[List Fn] -- This returns the first IsOk result of applying Fn to an element of List. (Note it breaks (i.e. gives a warning message) if any such element fails IsOk.)

MergeN[$List_1$ $List_2$ ... $List_N$] -- This is the *N*-ary form of the binary MERGE.

ReadYesNo[LiteralString] -- Prints out LiteralString as prompt, then requests input. Returns T if user responds with "Yes"; NIL otherswise.

SOS[new?] -- For debugging, we maintain a DRIBBLEFILE recording the activity of each session. Every uNUM-BETWEEN-SOS user responses, the function SOS-M, lying in PRCMPTCHARFORMS, calls on SOS to close and reopen the current dribble file, named DribbleName. In addition, if this is the uNUM-BETWEEN-SOS'th time this function has been called, it asks the user if he wishes to SYSOUT now.

If new? is nonNIL, SOS also finds a new name for the dribble file, before doing all the rest.

Setting uOKtoSOS to NIL turns off this dribbling.

SomeFn[List Fn] -- This is like SOME, except it returns the value of Fn(x) of the first element $x \in$ List which returns nonNIL, rather than the sublist of List which begins with x.

SubSetFn[List Fn] -- This is like SUBSET, except it returns the list of values, Fn(x), for each element $x \in$ List which returns nonNIL, rather than the sublist of List containing such x.

Tracer[FnName stk] -- This is an aide to debugging, especially useful when a recursive function seems to be in an infinite descent. It advises this function to maintain a stack, on the variable stk (defaulted to (PACK* FnName 'STK) if omitted) of outstanding arguments passed to this function.

UnionN[$List_1$ $List_2$ ... $List_N$] -- This is the *N*-ary form of the binary UNION.

WhatKB[Name kb must might-be-kb] -- This is used to find the knowledge base in which to enter the new thing, named "Name". If kb is supplied, and is a member of the UF.NETWORKS, that value is returned. Otherwise, if there is only one knowledge base open, that value will be returned. (If must is NIL, indicationg this new thing need not be affiliated, the user is first asked if this is appropriate. Otherwise, he is just informed that this affiliation has happened.) If nonNIL, the value of might-be-kb is suggested to the user, who can approve it by responding "Y".

WhichFnList[FnName] -- This returns the name of the list on which the function, FnName, is located. (Note the utility functions have been split up into several logically distinct classes, each with its own list. Also most knowledge bases come with their own such list of functions.)

WhoIsUser[val] -- This attempts to find (and return) the unit under AnyUser which represents val (which defaults to the current user). If none is found, and if allowed, it will create a new unit for this user, and return that value.

F.4: Advised Functions

Except where otherwise noted, every bit of this advice is set in Advisor, *and is permanent.*

LOGOUT is told to close the dribble file, and checks if the current state of this LISP session should be released -- i.e. if any writes onto a KB have been performed since the last SYSOUT.

As each function is defined, the user is asked if in which file he wishes to store this function. This KB management is done by the advise to \PUTD, PUTD, \MOVD and MOVD. LOAD, ADVISE, READVISE are all told to tell \PUTD's advice not to try to save the temporary functions they generate.

EDITE has been advised to do two things: First, it knows not to bother \PUTD's advise when a function's source code is read in. Secondly, it warns the user if he tries to use EDITP to edit a unit that his changes not only might not be saved, but he might really foul things up as well.

The only other function RLL-1 ever advises in UP-PUTUNIT. This is done by CleanWS, and is usually removed the next time UP-PUTUNIT is called. This advise tells the user some unit is about to written onto the external *.PAGE file; and permits him to enter readonly mode at this point.

F.5: Global Variables

PROMPTCHARFORMS has been changed, by adding (SCS-ME). This function helps maintain the dribble file.

AFTERSYSOUTFORMS, BEFORESYSOUTFORMS have been added to, so the right things happen when the user does a SYSOUT (i.e. *.PAGE files get saved, ...)

BeenStarted is T if the RLL-1 system has been started already, otherwise NIL.

DribbleName, uAllDribbleFiles are used by SOS, to record the current, and all previous dribble files used, respectively. uAllDribbleFiles is printed out by the advice to LOGOUT.

uNUM-BETWEEN-SOS stores a value SOS-ME uses to decide when it is time to call SOS. Currenlty set to 25. (SOS also uses it to decide when to suggest the user SYSOUT.)

uAllUsers stores all users who have used this sysout, and written out any changed units.

uSYSOUTNAME is the name of the last SYSOUT performed; or RLL-1.EXE if none have been done.

RenamedUnits, DeletedUnits store lists of units which have been renamed (as dotted pairs) or deleted, respectively.

WORRIES stores text describing past events which might later lead to trouble -- i.e. they were not serious enough to break on, but they still should not have happened.

KernelKB is the name of the starting, "top most" knowledge base -- currently, of course, RLL-1.

uValue, uContext are used when IMPURE is one of the values listed by others, when passed to DefaultGetValue. They are set to the value and unit on which this value was found, respectively. <uContext may be meaningless now.>

DefaultGetValueOptions is used by DefaultGetValue as the default value for others, when that argument is NIL.

G. Appendix - Sample Session

This appendix shows a session the author had with the current RLL-1 system. This is designed to provide a flavor of RLL-1's capabilities, as well as describe the formats for its various commands. Comments are shown enclosed in braces, "{}"s.

Table of Contents

<<<< 1. Starting the system >>>>

[I adviced USERNAME to return "NewUser" for this.]

@<CSD.IA>DEMO.EXE

Shall I start the RLL system now? *yes*

*** Am opening Dribble file: TRACE.Aug18 [18-Aug-80 13:38:45]

Reading in RLL.STATUS now.

Opening knowledge base <CSD.RLL>RLL.KB.12

Opening paging file <CSD.IA>RLL.PAGE.1

Loading unit RLL.UNITINDEX

Loading unit RLL.STATUS

Last written by (CSD.GREINER 16-Aug-80 22:18:15)

{{Loading unit KBsFNS
Loading unit AllIsas
Loading unit ToGetValue
Loading unit OrderedPrototypes
Loading unit Defn
Loading unit Prototypes
Loading unit MySlotsNowOrdered
Loading unit TypicalPrimSlot
Loading unit TypicalSlot
Loading unit BeforeGetValue
Loading unit AfterGetValue
Loading unit KBsVARS
Loading unit KBsConnectedTo}}

This kb, RLL is already connected to all of (RLL).

Do you wish to read in any Knowledge Bases? yes

{{Loading unit AllExamples
Loading unit AnyUser
Loading unit UserNames
Loading unit RussGreiner
Loading unit DougLenat
Loading unit LarryHines
Loading unit AndyFreeman}}

I don't know who you are!

Shall I create a unit to store information about you? yes

Is NewUser an appropriate name? no

What name would you prefer? Visitor

{{Loading unit GetPossibleSlotsFn
Loading unit IExamples
Loading unit LispFn
Loading unit TypicalVirtualSlot
Loading unit PossibleSlotsOfIExamples

```
Loading unit StoredAList
Loading unit GenlsModels
Loading unit SuperClass*
Loading unit TypicalExample
Loading unit Anything
Loading unit AnyCT&U
Loading unit NewPossibleSlots
Loading unit TypicalCT&U
Loading unit MyCreator
Loading unit MyTimeOfCreation
Loading unit MyToKillMe
Loading unit TypicalAccessSlot
Loading unit MyToRenameMe
Loading unit MyEssentialVirtualSlots
Loading unit MySensibleSlots
Loading unit MyCreatedAs
Loading unit MySlots
Loading unit TypicalThing
Loading unit Specializations
Loading unit TypicalUnitFn
Loading unit TypicalStorableFn
Loading unit TypicalFunction
Loading unit TypicalProcess
Loading unit Isa
Loading unit AllGenls
Loading unit AllSpecs
Loading unit Characteristics
Loading unit Descr
Loading unit TypicalUser
Loading unit InformalName
Loading unit UsualKBs
Loading unit WritingOptions
Loading unit OpenningOptions
Loading unit OrderForToInit
Loading unit Typical$SELF$Slot
Loading unit ToAddValue
Loading unit BeforePutValue
Loading unit VerifyElement
Loading unit Format
Loading unit FnForPutting
Loading unit FSet
Loading unit SuperTypEx*
Loading unit TypicalExampleOf
Loading unit SuperClass
Loading unit FnForAdding
Loading unit ToCache
Loading unit ToPutValue
Loading unit AfterPutValue
Loading unit Inverse
Loading unit KBUpdates
Loading unit Examples
Loading unit AllTypicalExampleOfs})

  1 -- RESOL
  2 -- GENLINFO
  3 -- BIOLOGY
  4 -- SETS
  5 -- MATH
  6 -- NUMBER
  7 -- HOBBIT
```

```
    8 -- HEURS
    9 -- OLD
    10 -- EURISKO

  Enter the numbers of the ones you wish to use: 2

  {{Loading unit FunctionSpec
    Loading unit HighLevelDefn
    Loading unit DomainType
    Loading unit FList}}

  Opening knowledge base <CSD.RLL>GENLINFO.KB.4

  Opening paging file <CSD.IA>GENLINFO.PAGE.1

  {{Loading unit GENLINFO.UNITINDEX
    Loading unit GENLINFO.STATUS}}

  Last written by (CSD.HINES 31-Jul-80 15:38:24)

  Network RLL already open.

  This kb, GENLINFO is already connected to all of (GENLINFO RLL).

  KBs loaded.

  (<CSD.IA>DEMO.EXE.3 . <LISP>LISP.EXE.133)

                          <<<<      2. What was created?    >>>>

       {Turn off those obnoxious Loading, ... messages}

  99-(SETQ UP.TRACEFILE NIL]
  (UP.TRACEFILE reset)
  NIL

       {Now look at what was just created}

  100-EDITU(Visitor]
  edit

  98*pp
       {Note this is a property list, of the form (slot1 value1 slot1 value2 ...}

  (Isa (AnyUser)
  UserNames ("NewUser")
  AllIsas (AnyCT&U Anything AnyUser)
  Prototypes (TypicalUser TypicalCT&U TypicalThing)
  MySlotsNowOrdered (OrderedPrototypes)
       {NOTE: Slots beginning with "My" are syntactic.}
  UsualKBs (RLL)
  MyCreatedAs (IExamples (AnyUser))
  MyTimeOfCreation "18-Aug-80 13:40:23"
  MyCreator "NewUser"
  OpenningOptions NoEntries
  MySensibleSlots (Descr Characteristics Prototypes AllSpecs AllGenls AllIsas Isa
            OrderedPrototypes Specializations MySlots MyCreatedAs
            MySensibleSlots MySlotsNowOrdered MyEssentialVirtualSlots
            MyToRenameMe MyToKillMe MyTimeOfCreation MyCreator UserNames
            OpenningOptions WritingOptions UsualKBs InformalName))
```

98*ok
Nothing changed.
Visitor

<<<< 3. Looking around >>>>

{Let's see the top level units:}

1←DI(Anything S 2]

Note the trace flag has been turned off.

Anything
AnyAT&U
AnyAbstractThing
AnyCT&U
AnyConcreteThing
AnyUnit

!DONE!

{This showed the subclasses of Anything, down (a total of) two levels
The ...&U units are hacks, to store both the unit and its meta-unit
in the same physical unit.

More interesting is:}

2←DI(AnyCT&U S 2]

Note the trace flag has been turned off.

AnyCT&U
AnyClassOfObjects
AnyDatatype
AnyDecomposableObject
AnyFormat
AnyInheritance
AnyIntensionalObject
AnyOverhead
AnyProcess
AnyUnit
AnyUser

!DONE!

<<<< 4. Creating a new class >>>>

{Let us create a new unit - which refers to people:}
4←NewSpec(AnyPerson]
Is a ISubClass of: AnyCT&U
Please enter the Knowledge Base in which to store AnyPerson: GENLINFO

Is the format of (LAMBDA (units sl? oth?) (MapUnion (IsOk units)
(FUNCTION (LAMBDA (x) (GetValue x (QUOTE Prototypes) (AddOnCharacter
oth? (QUOTE VERYSAFESLOT)))))))) a list?
yes

{Here, RLL is asking about the Format of the expression (LAMBDA ...).

```
        This was because it didn't know about the function "MapUnion".
        It will, later in RLL's development.}

   You are about to write on an external file.

   Do you want to enter ReadOnly mode? no

        {Before the first write, RLL gives the user a chance to leave
        the systems unmodified. Here, we told RLL to go ahead.}
    * Initialized AnyPerson *
   edit

   99*p

   (Isa (AnyClassOfObjects)
   AllIsas (AnyCT&U Anything AnyClassOfObjects)
   Prototypes (TypicalClass TypicalCT&U TypicalThing)
   MySlotsNowOrdered (OrderedPrototypes) _
   MyCreatedAs (ISubClass &)
   MyTimeOfCreation "18-Aug-80 13:50:28"
   MyCreator "NewUser"
   TotalSoFar 0
   SuperClass (AnyCT&U)
   MySensibleSlots (Descr Characteristics Prototypes AllSpecs AllGenls
              AllIsas Isa OrderedPrototypes Specializations MySlots MyCreatedAs
              MySensibleSlots MySlotsNowOrdered  MyEssentialVirtualSlots
              MyToRenameMe MyToKillMe MyTimeOfCreation MyCreator TotalSoFar --))
        {  ↑ These are slots which are well defined for this unit.}

    *** Am re-opening Dribble file: TRACE.Aug18 [18-Aug-80 13:53:01]

        {↑ A dribble file records the session. (You're looking at it now.]
        { This class unit looks }
   100*ok
   Verifying slots
   AnyPerson
```

<<<< 5. Creating a new typical example, for a class >>>>

```
        {Now we'll create a unit which hold facts typically true of any person.
        (That is, default values; as well as a list of new slots to be inherited
        by every new person.]}

   5-NewTypEx]
   Name: TypicalPerson
   Is a ITypEx of: AnyPerson
   Please enter the Knowledge Base in which to store TypicalPerson: GENLINFO

   Is the format of (LAMBDA (units sI? oth?) (MapUnion (IsOk units) (FUNCTION (LAMBDA
   (uNITcomp sLOTcomp oTHERcomp) (OR (MapUnion (GetValue uNITcomp (QUOTE GenlsModels)
   (AddOnCharacter oTHERcomp (QUOTE VERYSAFESLOT)))
   (LAMBDA (x) (GetValue x (QUOTE Prototypes) (AddOnCharacter oTHERcomp
   (QUOTE VERYSAFESLOT)))))) NoEntries)))))
   a list?
   yes

        {No, RLL still doesn't know about MapUnion.}
   Shall I create a slot with the high level defn:
   (Composition TypicalExample SubClass*)?  no
```

```
        {To update inverse links, it has to "invert" each relevant slot.
         For this it uses that slot's high level definition... which goes to
         the slot combiners involved, ...
         Above it was inverting GenlsModels, whose HighLevelDefn was
         (Composition SuperClass* TypicalExampleOf]

         RLL was asking whether to preserve this definition as or slot, or not.
         I said NO.}
 * Initialized TypicalPerson *
edit

1*p

(Isa (AnyArchetype)
 TypicalExampleOf AnyPerson
 NewPossibleSlots NoEntries
 AllIsas (AnyIntensionalObject AnyCT&U Anything AnyAT&U AnyArchetype)
 Prototypes (TypicalTypicalEx TypicalCT&U TypicalAT&U TypicalThing)
 MySlotsNowOrdered (OrderedPrototypes)
 MyCreatedAs (ITypEx &)
 MyTimeOfCreation "18-Aug-80 13:59:43"
 MyCreator "NewUser"
 MySensibleSlots (Descr Characteristics Prototypes AllSpecs
                  AllGenls AllIsas Isa OrderedPrototypes Specializations MySlots
                  MyCreatedAs MySensibleSlots MySlotsNowOrdered
                  MyEssentialVirtualSlots MyToRenameMe MyToKillMe MyTimeOfCreation
                  MyCreator SubTypEx* SuperTypEx* SubTypEx SuperTypEx
                  TypicalExampleOf NewPossibleSlots PossibleSlots))

       {↑ These slots are pertanent to all prototypes, such as this TypicalPerson.
        RLL determined these using from the prototypical prototype,
        TypicalTypicalEx.}
2*5 up p
... NewPossibleSlots NoEntries

       {↑ its value lists the new slots, which all people will have.}

       {Currently it is empty.
        (Note we use "NoEntries" to indicate a list we know is empty,
        leaving NIL to mean a value we simply don't know.]}
3*(2 (Mother Husband))

       {Now new examples of AnyPerson will have these two slots.}
4*bk (b Mother NoEntry Husband NoEntry]

5*p

... Mother NoEntry Husband NoEntry NewPossibleSlots (Mother Husband) ...

       {(NoEntry is like NoEntries in purpose. It, however,
        refers to the absense of a single entry.]

        Here, it is used to indicate that TypicalPerson cannot
        provide any sort of default information about a new unit's relatives.

       It also causes RLL to consider creating a Mother (resp. Husband)
        type of slot. This is done in EDITU.}
6*ok
```

<<<< 6. Slot verification by EDITU >>>>

```
Verifying slots
Your attempted slot name Mother is NOT even a unit. Should it be? yes
      {This will create a new unit, to house facts about the "Mother" slot.}
What should the Isa link for this Mother link be?
      {It will be created as a new Primitive slot - hence the "P" response above.}

Expecting one of:
(P V A *)

What should the Isa link for this Mother link be? P
Please enter the Knowledge Base in which to store Mother: GENLINFO
 * Initialized Mother *
edit
8*pp
(Isa (PrimSlot)
 AllIsas (AnySlot AnyUnitListFn AnyFunction AnyCT&U Anything AnyProcess
       AnyStorableFn AnyUnitFunction PrimSlot)
 Prototypes (TypicalPrimSlot TypicalSlot TypicalUnitFn TypicalStorableFn
       TypicalFunction TypicalProcess TypicalCT&U TypicalThing)
 MySlotsNowOrdered (OrderedPrototypes)
 MyCreatedAs (IExamples (PrimSlot))
 MyTimeOfCreation "18-Aug-80 14:10:56"
 MyCreator "NewUser"
 Format (*Do* FOneOf FSingleton FSet FList FOrderedSet FBag)
      {We'll see soon this special value, ↑, will be useful.}
 Datatype (NonNILType)
 MySensibleSlots (Descr Characteristics Prototypes AllSpecs AllGenls AllIsas Isa
            OrderedPrototypes Specializations MySlots MyCreatedAs
            MySensibleSlots MySlotsNowOrdered MyEssentialVirtualSlots
            MyToRenameMe MyToKillMe MyTimeOfCreation MyCreator WhatToProcess
            HowToProcess LispFn IUseDefnOf IUseCVOf DefnUsedBy CVUsedBy
            FunctionCharacter RangeType DomainType PreConditions Range Domain
            Definition FunctionSpec Format Datatype DataRange Defn
            HighLevelDefn IsBuiltFrom UsingFunctions SlotsBuiltFrom
            UsingFunctionals UnitsBuiltFrom SlotsUsedInBuilding ToLookUp
            ToCache LispFnForStoredFn StoredAList ToConfirmValue
            UsingSlotCombiners ToGetValue HandDoneSBF AllSBF VerifyElement
            VerifyAll ToSubstValue ToDeleteValue ToAddValue MakesSenseFor
            SubSlot* SuperSlot* SubSlot SuperSlot ToInitialize ToPutValue
            OrderForToInit KBUpdates Inverse IsEssentialFor))
8*f Datatype
9*p
... Datatype (NonNILType) ...
9*(2 (UnitType) DataRange (*P AnyPerson]
      {This "UnitType" indicates the value of U:Mother will be a unit;
       and the DataRange restricts that, to say that unit will descend
       from AnyPerson}
10*↑ f Format
12*2 p
(*Do* FOneOf FSingleton FSet FList FOrderedSet FBag)
      {IE The format must be one of these.
       We'll see soon that each of these formats is really a unit.}
13*xtr 3
      {This means U:Mother will be filled with a single entry.
       That's all I want to say about Mothers, at this time.}
14*ok
Verifying slots
      {We now pop back to editing TypicalPerson, and find another non-slot:}
```

```
Your attempted slot name Husband is NOT even a unit. Should it be? yes
What should the Isa link for this Husband link be? P
Please enter the Knowledge Base in which to store Husband: GENLINFO
 " Initialized Husband "
edit
      {To fix Husband's range specification:}
15*f Datatype
16*2 p
(NonNILType)
17*: (UnitType) DataRange (*P AnyPersen]
18*ok
Verifying slots
What should the value of Husband:Format be?


Expecting one of:
(FSingleton FSet FList FOrderedSet FBag)

What should the value of Husband:Format be? FSingleton
TypicalPerson
      {Note it asked for Husband's format, as I hadn't specified it; and RLL
       figured it would be needed eventually.}

              <<<<      7. Examining unit which represent Formats    >>>>
      {Note first that there are two subclasses of formats:}
30-SubClass(AnyFormat]
(AnySlotFormat AnyValueFormat)

      {The interesting one is}
31-Examples(AnySlotFormat]
(FSingleton FList FSet FOrderedSet FBag FListN)

      {Each of these is a bonafide unit:}
32-EDITU(FSet]
edit
66*p
(Isa (AnySlotFormat)
 FnForVerifyingAll (LAMBDA & &)
 FnForVerifyingElement (LAMBDA & &)
 FormatCharacter (MayBeEmpty NonOrdered NoDuplicated ArbitraryNumberOfEntries)
 FnForAdding (LAMBDA & &)
 FnForDeleting (LAMBDA & & &)
 FnForSubstituting (LAMBDA & & &)
 FnForPutting (LAMBDA & &) --)
      {The value for each of the FnFor--- is a function, which is used by some
       accessing function (eg GetValue) to view or alter a value.}

66*ok
Nothing changed.
FSet

36-(MAPCAR (Examples 'AnySlotFormat) 'FormatCharacter]

((MayBeEmpty SingleEntry)
 (MayBeEmpty Ordered Duplicates ArbitraryNumberOfEntries)
 (MayBeEmpty NonOrdered NoDuplicated ArbitraryNumberOfEntries)
 (MayBeEmpty Ordered NoDuplicated ArbitraryNumberOfEntries)
 (MayBeEmpty NonOrdered Duplicates ArbitraryNumberOfEntries)
 (MayBeEmpty Ordered Duplicates ExactNumberOfEntries))
```

```
                    <<<<    8. Adding on a new person    >>>>

      {Back to our main plot:
       Let's make Visitor think of itself as a person.}
6←EDITU(Visitor]
edit
19*p
(Isa (AnyUser) UserNames ("NewUser")  ... )

19*2 (n AnyPerson]

21*0 p

(Isa (AnyUser AnyPerson) UserNames ("NewUser") ... )

22*ok
Verifying slots
Visitor

      {Note I could have made AnyUser a subclass of AnyPerson,
       but who knows who (or what) will eventually be using RLL...}
7←EDITU]
=Visitor
edit
      {Let's give our visitor a mother:}

23*(n Mother HisMother]
      {(Note this should NOT work, as we have yet to define "HisMother"...)}
28*ok
Verifying slots

Trouble doing actual Put (Visitor Mother HisMother)
Shall I go on, break, or edit this value? Go

Visitor

                    <<<<    9. Creating a new entity - HisMother    >>>>
      {So let's create that unit.}
8←NewIsa)
Name: HisMother
Is a IExamples of: AnyPerson
Please enter the Knowledge Base in which to store HisMother: GENLINFO
 * Initialized HisMother *
edit
29*pp

(Isa (AnyPerson)
 AllIsas (AnyCT&U Anything AnyPerson)
 Prototypes (TypicalPerson TypicalCT&U TypicalThing)
 MySlotsNowOrdered (OrderedPrototypes)
 MyCreatedAs (IExamples (AnyPerson))
 MyTimeOfCreation "18-Aug-80 14:27:04"
 MyCreator "NewUser"
 MySensibleSlots (Descr Characteristics Prototypes AllSpecs AllGenls AllIsas Isa
          OrderedPrototypes Specializations MySlots MyCreatedAs
          MySensibleSlots MySlotsNowOrderod MyEssentialVirtualSlots
          MyToRenameMe MyToKillMe MyTimeOfCreation MyCreator Husband Mother))

29*ok
Verifying slots
```

```
HisMother

                    <<<<    10. How are new units created?    >>>>

37-PP(NewIsa]
loading from <CSD.RLL>UTIL..7

(NewIsa
  [LAMBDA (Son Parent whichKB) **COMMENT**   **COMMENT**
    (NewUnit Son Parent (QUOTE IExamples) whichKB T])
(NewIsa)

      {So what is "IExamples}

38-EDITU(IExamples]
edit
69*p
(Isa (AnyInheritance)
 Descr (Here is a the Inheritance fo creating something which is an examples
       of some class.)
 UseToGetSlots GenlsModels
 GetPossibleSlotsFn  PossibleSlotsOfIExamples)

      {There may eventually be other things stored here, once we figure out
       what we mean when we say "Inheritance".}
69*ok
Nothing changed.
IExamples

      {Are there other inheritances?}
39-AllExamples(AnyInheritance]
(ITypEx ISubClass IExamples)

      {What is that PossibleSlotsOfIExamples?}
39-EDITU(PossibleSlotsOfIExamples]
edit
70*p
(UsedByInheritance IExamples
 Isa (AnySlotGetter)
 Format FOrderedSet
 HighLevelDefn (PutInOrder (CommonXProd NewPossibleSlots
                                (ApplyToEach MapUnion GenlsModels)
                                WhereInitFn)
                           OrderForToInit CAR NIL)
 ... )
      {Wow - look at that HighLevelDefn!}
72*ok
Nothing changed.
PossibleSlotsOfIExamples

                    <<<<    11. Hack to LISP's evaluator    >>>>
      {To see our collection of people:}
9-Examples(AnyPerson]
(HisMother Visitor)

      ( 1) That is, as U:Isa was filled with AnyPerson,
           U was added to AnyPerson:Examples
        2) Note there is no function named "Examples".
           That call worked because we've hacked up LISP's evaluator to try
           (GetValue U 'S) if S(U) fails)
```

```
10←GETD(Examples]
NIL

      {It is, of course, a unit:}
11←Unitp(Examples]
RLL
      {The same is true for AllExamples, shown last page.}

                <<<<      12. Creating a new datatype - GenderType    >>>>

      {Lets now create a new datatype, to help us distinguish
      Males from Females
      First, what are the current datatypes?}
11←Examples(AnyDatatype]
(KBType TVType NonNILType GrammarType NumberType BooleanType UnrestrictedType
      FunctionType SlotType IntegerType UnitType StringType)

      {We'll create a copy of BooleanType, then modify that copy,
       using the function}
12←NU(GenderType)
Copy from: BooleanType
Please enter the Knowledge Base in which to store GenderType: GENLINFO
edit

30*pp
(Isa (AnyDatatype)
 VerifyType [LAMBDA (x)
          (FMEMB x (QUOTE (T F]
 GenerateAll [LAMBDA NIL (QUOTE (T F]
 IsTypeOf NoEntries
 SuperDT (NonNILType)
 MyCreatedAs (IExamples (AnyDatatype)))
      {Apparently the function stored in VerifyType returns nonNIL
       if its argument qualifies as a member of this datatype.
       (We could have found this out by looking on the "VerifyType" unit...)
       and GenerateAll returns a list of these acceptable values.}
30*(R (T F) (Male Female Neuter Hermaphroditic]

34*PP
(Isa (AnyDatatype)
 VerifyType [LAMBDA (x)
          (FMEMB x (QUOTE (Male Female Neuter Hermaphroditic]
 GenerateAll [LAMBDA NIL (QUOTE (Male Female Neuter Hermaphroditic]
 IsTypeOf NoEntries
 ...
      {Everything else looks right. so}
42*ok
Verifying slots
GenderType

      {Note another way of doing this would be to create a new class, AnyGender
       and have a unit for each of these - Male, Female, ...
       This was how Formats, Datatypes, Inheritances, ...
       were all handled.
      Let's now have genders, all around}

                <<<<      13. Engendering our visitor    >>>>

13←EDITU(Visitor]
```

```
edit
43*(-1 Gender Male]

      {Showing my prejudices, I'll assume this visitor is masculine.
       Note there is now no unit named Gender. RLL will notice that also.}
44*p
(Gender Male Isa (AnyUser AnyPerson) UserNames ("NewUser") ... )

44*ok
Verifying slots

Your attempted slot name Gender is NOT even a unit. Should it be? yes
What should the Isa link for this Gender link be? P
Please enter the Knowledge Base in which to store Gender: GENLINFO

 * Initialized Gender *

edit
45*p
(Isa (PrimSlot) AllIsas (AnySlot AnyUnitListFn AnyFunction AnyCT&U Anything
AnyProcess AnyStorableFn AnyUnitFunction PrimSlot) Prototypes (TypicalPrimSlot
TypicalSlot TypicalUnitFn TypicalStorableFn TypicalFunction TypicalProcess
TypicalCT&U TypicalThing) MySlotsNowOrdered (OrderedPrototypes) MyCreatedAs (
lExamples &) MyTimeOfCreation "18-Aug-80 14:43:27" MyCreator "NewUser" Format (
*Do* FOneOf FSingleton FSet FList FOrderedSet FBag) Datatype (NonNILType)
--)

      {Lets fix up the value of Datatype, using our newly created one:}
45*15 up p
... Datatype (NonNILType) MySensibleSlots (Descr Characteristics  ... ))

47*(2 (GenderType]

      {I'll also indicate that only people can have genders, using the
      MakesSenseFor slot:}
49*(n MakesSenseFor (TypicalPerson]
50*ok
Verifying slots

What should the value of Gender:Format be? FSingleton
Visitor

      {As Inverse(MakesSenseFor) is NewPossibleSlots,
       Let's now look at the NewPossibleSlots of TypicalPerson}

46←NewPossibleSlots(TypicalPerson]
(Gender Mother Husband)

                <<<<      14. Indicating that Mothers are female    >>>>

      {Let's now specify that all Mothers are female:
       First, how to say that range specification,
       look at the UnitType datatype.}
47←EDITU(UnitType]
edit
93*p

(Isa (AnyDatatype)
 VerifyType Unitp
 IsTypeOf (Husband KBsConnectedTo Isa NewPossibleSlots  ...)
```

```
 RangeInterpreter UnitRange
 SuperDT (NonNILType)
 SubDT (SlotType)
 MyCreatedAs (IExamples &)
 AllIsas (AnyCT&U Anything AnyDatatype))

93*f RangeInterpreter
94*p

... RangeInterpreter UnitRange ...

      {The value of DT:RangeInterpreter is a function, which uses a slot's
       DataRange value to compose a function. That function is used to
       restrict the values acceptable for that slot.}
94*2 p
UnitRange

      {Lets look at this functions}
95*ef
loading from <CSD.RLL>RLL..6
prop
edit
95*p

(LAMBDA (rangespec valname)  **COMMENT**  (AND & &))

      {Details omitted for brevity.}
95*ok
not changed, so not unsaved
UnitRange

96*ok
Nothing changed.
UnitType

      {What is the current range type of Mother?}
48←RangeType(Mother]
[FSingleton (UnitType (*P AnyPerson)]

      {That is, the value of U:Mother is a single value, which is a unit,
       which descends from AnyPerson.
       Now to add that specification to Mother:}

49←EDITU(Mother]
edit

97*f DataRange
98*2 p
(*P AnyPerson)
99*mbd (L-AND * (SlotVal Gender Female]
100*p
(L-AND (*P AnyPerson) (SlotVal Gender Female))

      {By the way, there is a unit for this L-AND}
100*EU
edit

100*p

(Isa (AnyLogicalOp)
```

```
 MyCreator "CSD.GREINER"
 MyTimeOfCreation "15-Apr-80 17:43:55"
 MyCreatedAs (IExamples &)
 Defn (LAMBDA & &))

100*ok
Nothing changed.
L-AND

1*ok
Verifying slots
Mother

      {Now to show that Mother's RangeType has changed:}
51←RangeType(Mother]

[FSingleton (UnitType (L-AND (*P AnyPerson)
                     (SlotVal Gender Female]

                        <<<<      15. Add another person    >>>>

      {Add another person to cur KB}

52←NewIsa(Fred AnyPerson GENLINFO]
 * Initialized Fred *
edit

2*p

(Isa (AnyPerson) AllIsas (AnyCT&U Anything AnyPerson) Prototypes (TypicalPerson
TypicalCT&U TypicalThing) MySlotsNowOrdered (OrderedPrototypes) MyCreatedAs (
IExamples &) MyTimeOfCreation "18-Aug-80 15:03:25" MyCreator "NewUser"
MySensibleSlots (Descr Characteristics Prototypes AllSpecs AllGenls AllIsas Isa
OrderedPrototypes Specializations MySlots MyCreatedAs MySensibleSlots
MySlotsNowOrdered MyEssentialVirtualSlots MyToRenameMe MyToKillMe
MyTimeOfCreation MyCreator Husband --))

      {Lets engender Fred}
2*(n Gender Male]
ok
Verifying slots
Fred

      {As a side comment, creating Fred took much less time than creating
       HisMother, as much of the information needed to create a new example
       AnyPerson was cached away during this first computation.
      This was true in general - the first time a new unit is created using
       inheritance I, from the list of parents, P, it will take a long time.
       Subsequent children of these parents will created and initialized much
       faster.}

                        <<<<      16. Can a male be a mother?    >>>>

     {Lets see if we're allowed to make Fred a mother:
      (Note PutValue returns NIL only if some error had been encountered.]}
53←PutValue(Fred Mother Visitor]
NIL

     {That is, no value was put -- i.e.}
```

```
68←UA-GETVALUE(Fred Mother]
NIL

      {[Note UA-GETVALUE is like GETPROP - no smarts]
       We now show PutValue can do something:
       First, let's make HisMother Female:}
69←PutValue(HisMother Gender Female]
Female

70←PutValue(Fred Mother HisMother]
HisMother

      {Proof:}
71←UA-GETVALUE(Fred Mother]
HisMother
```

<<<< 17. Now make Husbands male: >>>>

```
73←EDITU(Husband]
edit

19*f DataRange

20*p

... DataRange (*P AnyPerson) ...

20*2 MBD (L-AND * (SlotVal Gender Male]

21*1 p

... (L-AND (*P AnyPerson) (SlotVal Gender Male)) ...

22*ok

Verifying slots
Husband

      {Just to check}

75←RangeType(Husband]
[FSingleton (UnitType (L-AND (*P AnyPerson)
                     (SlotVal Gender Male]
```

<<<< 18. To give HisMother a Husband >>>>

```
76←NewIsa)
Name: HerHusband
Is a !Examples of: AnyPerson
Please enter the Knowledge Base in which to store HerHusband: GENLINFO
 * Initialized HerHusband *

edit
23*pp
(Isa (AnyPerson)
 AllIsas (AnyCT&U Anything AnyPerson)
 Prototypes (TypicalPerson TypicalCT&U TypicalThing)
 MySlotsNowOrdered (OrderedPrototypes)
```

```
 MyCreatedAs (IExamples (AnyPerson))
 MyTimeOfCreation "18-Aug-80 15:15:50"
 MyCreator "NewUser"
 MySensibleSlots (Descr Characteristics Prototypes AllSpecs AllGenls AllIsas Isa
          OrderedPrototypes Specializations MySlots MyCreatedAs
          MySensibleSlots MySlotsNowOrdered MyEssentialVirtualSlots
          MyToRenameMe MyToKillMe MyTimeOfCreation MyCreator Husband Mother
          Gender))

     {Note Gender is on MySensibleSlots now
      (of course it wasn't before it existed.]}

23*(n Gender Male]
24*ok

Verifying slots
HerHusband

77←PutValue(HisMother Husband HerHusband]
HerHusband
     {Now HisMother:Husband is HerHusband, as planned.}

                <<<<     19. Create a new type of slot - Father    >>>>

78←SMARTARGLIST(CreateSlot]
(hid kb extra-pairs name)

     {To define Father}
78←(CreateSlot '(Composition Husband Mother) 'GENLINFO NIL 'Father]
Father

     {To see if it worked:}
79←Father(Fred]
HerHusband

     {Lets see what this unit really looks like}

88←EDITU(Father]
edit
36*F Defn
37*2 pp

[LAMBDA (uNITcomp sLOTcomp oTHERcomp)
  (OR (GetValue (GetValue uNITcomp (QUOTE Mother)
                (AddOnCharacter oTHERcomp (QUOTE VERYSAFESLOT)))
          (QUOTE Husband)
          (AddOnCharacter oTHERcomp (QUOTE VERYSAFESLOT)))
    NoEntry]
38*ok
Nothing changed.
Father

                <<<<     20. There is a unit for Composition:    >>>>

26←EDITU(Composition]
edit
64*pp
(Isa (AnySlotCombiner)
 Descr (Compose S1 of S2 of ... of Sn the unit)
```

```
FnForCaching NoEntry
FnForUpdating [LAMBDA (affectedsit fullHLD changedsit argnames hold)
  (OR [MAPCONC (REVERSE (CDR fullHLD))
          (FUNCTION (LAMBDA (sit morework)
             (PROG1 (COND
                    [(LISTP sit)
                     (COND
                      [hold (AND (SETQ morework
                                  (UpdateASUIB
                                   sit changedsit
                                   (CONS (QUOTE x)
                                         (CDR argnames))
                                   affectedsit))
                                 (DoToEachFn
                                  (COND
                                   ((CDR hold)
                                    (CONS (QUOTE Composition)
                                          hold))
                                   (T (CAR hold)))
                                  changedsit argnames
                                  (ConsN (QUOTE LAMBDA)
                                         (QUOTE (x))
                                         morework]
                      (T (UpdateASUIB sit changedsit argnames
                                      affectedsit]
                     [(EQ sit changedsit)
                      (COND
                       (hold (InvalidateInverseFn
                              (COND
                               ((CDR hold)
                                (CONS (QUOTE Composition)
                                      hold))
                               (T (CAR hold)))
                              affectedsit changedsit argnames))
                       (T (InvalidateFn affectedsit changedsit
                                        argnames]
                     (T NIL))
                    (SETQ hold (CONS sit hold]
          (LIST NIL]
CombinerFor (PossibleSlotsOfITypEx AllIsas AllExamples SuperTypEx* SubTypEx*)
Defn [LAMBDA (slotlist args)
  (SETQ slotlist ([LAMBDA (**SELF**)
      (APPLY* (GetValue (QUOTE Composition)
                        (QUOTE ToParseParts)
                        (QUOTE (VERYSAFESLOT VERYSAFEUNIT)))
              slotlist]
    NIL))
  (PROG (walker answer isLIST)
      (SETQ walker (REVERSE slotlist))
      (SETQ answer (GetGetVal (CAR walker)
                              (QUOTE uNITcomp)
                              (QUOTE oTHERcomp)))
      [SETQ isLIST (ListFormat (HardFormat (CAR walker]
    LOOP(COND
        ((CDR walker)
         (SETQ walker (CDR walker))
         (SETQ answer (ComposeAux (CAR walker)
                                  answer
                                  (QUOTE oTHERcomp)))
         (GO LOOP))
```

```
        (T (RETURN (LIST [LIST (QUOTE LAMBDA)
                               (QUOTE (uNITcomp sLOTcomp oTHERcomp))
                               (LIST (QUOTE OR)
                                     (IsOk answer)
                                     (COND
                                       (IsLIST (QUOTE NoEntries))
                                       (T (QUOTE NoEntry]
                         [CONS (COND
                                 (isLIST (QUOTE FSet))
                                 (T (QUOTE FSingleton)))
                               (CDR (RangeTypeOf (CAR slotlist]
                         (DomainTypeOf (CAR (LAST slotlist)))
                         (QUOTE PSEUDO-SLOT]
FnForInverting [LAMBDA (hldefn temp)
  (AND [EVERY (CDR hldefn)
         (FUNCTION (LAMBDA (x pmet)
           (AND (SETQ pmet (InvertHLD x))
                (SETQ temp (CONS pmet temp]
     (CONS (QUOTE Composition)
           temp]
RangeType (FSingleton SlotType)
GetFnsUsed [LAMBDA (hld sc)
  (CONS (CAR hld)
        (MapUnion (CDR hld)
                  (FUNCTION (LAMBDA (term)
                    (AND (LISTP term)
                         (GetAllFNS term]
DefnUsedBy (AllIsas)
GetCVsUsed [LAMBDA (hld sc)
  (OR [MapUnion (CDR hld)
                (FUNCTION (LAMBDA (term)
                  (COND
                    ((ATOM term)
                     (LIST term))
                    (T (GetAllCVs term]
      NoEntries]
 AllIsas (AnySlotListFn AnyStorableFn AnyProcess Anything AnyCT&U AnyFunction
                  AnyUnitListFn AnyFunctional AnySlotCombiner)
 Prototypes (TypicalSlotCombiner TypicalSlotListFn TypicalStorableFn
          TypicalFunctional TypicalFunction TypicalProcess TypicalCT&U
          TypicalThing)
 MySlotsNowOrdered (OrderedPrototypes)
 ToParseParts (LAMBDA (args) (MAPCAR args (FUNCTION HLDefnParser]
64*ok
Nothing changed.
Composition

                    <<<<      21. Examples of SlotCombiners    >>>>

     {There are several exising slot combiners:}
6←(AllExamples 'AnySlotCombiner]
(Subsetting CommonXProd Intersecting FirstOk Starring Application
        Composition Tripple OrderedStarring OrderedComposition
        Unioning OrderedUnioning Plussing OneOf Soften PutInOrder
        Listing DoneIndirectly)

     {This class belongs in a subcategory of two more general classes:}
7←SuperClass(AnySlotCombiner]
(AnySlotListFn AnyFunctional)
```

{The first contains units which represent functions which each take a
list of slots as its argument.
It, in turn is a subclass of AnyUnitListFunction - i.e. functions
which take a list of units as arguments.
The second contains those "functions" which return, as a value,
another function. This category also includes, for example,
logical-operators, which map a list of predicates into a new predicate,
which returns T, e.g., when each of those predicates would return T.
Appendix C shows these classes in more detail.}

<<<< 22. What else can we say about Father? >>>>

```
90←RangeType(Father]

[FSingleton (UnitType (L-AND (*P AnyPerson)
                    (SlotVal Gender Male]
```

{Let's see if we're allowed to say "HisMother" is someone's Father}

```
91←(PutValue 'Visitor 'Father 'HisMother]
NIL
```

<<<< 23. Far too quiet: >>>>

Isn't that annoying? It would especially even more so if we didn't know
why this function failed.
(Here, it's because Father must be Male, and "HisMother" isn't.]
Let's make it noisier -- in fact, let's create a whole new class of slots,
which are more informative.
To see how it will work, let us first see how PutValue really works:

<<<< 24. How do accessing functions really work? >>>>

```
95←PP(PutValue]
loading from <CSD.RLL>UTIL..7

(PutValue
  [LAMBDA (uNIT sLOT Val old why) **COMMENT**   **COMMENT**   **COMMENT**
    (APPLY* (GetAccessFn sLOT (QUOTE ToPutValue)
                    (QUOTE (VERYSAFESLOT))
                    (QUOTE UA-DELSLOT))
          uNIT sLOT Val old why])
(PutValue)
```

{So you see, it basically just goes to the slot, and asks it how
put a value.
It applies the result of that GetAccessFn call on its list of arguments}

```
96←PP(GetAccessFn]
loading from <CSD.RLL>RLL..6

(GetAccessFn
  [LAMBDA (sLOT thisslot oTHER dftfn) **COMMENT**   **COMMENT**
    (OR (MEMB (QUOTE IMPURE)
              oTHER)
        (SETQ uContext sLOT))
    (SET (COND
          ((MEMB (QUOTE IMPURE)
                 oTHER)
           (QUOTE uValue))
          (T (QUOTE oTHER)))
```

```
      (OR (IsOk (UA~GETVALUE sLOT thisslot))
          (IsOk (FindDefault sLOT thisslot oTHER))
          (CheckDefn (Warning (CONCAT "Unable to find the " thisslot
                                      " slot of "
                                      sLOT ". Perhaps it is not a unit?")))
          dftfn
          (QUOTE NoOp])
(GetAccessFn)}}
```

{Essentially this sees if there is a value stored on sLOT:thisslot
If so, it uses that value. Otherwise, it calls FindDefault, which}

<<<< 25. The FindDefault function: >>>>

```
97~PP(FindDefault]
loading from <CSD.RLL>RLL..6

(FindDefault
 [LAMBDA (uUNIT uSLOT oTHERs) **COMMENT**
   (AND (Slotp uSLOT)
        (MapUntilOk [GetValue uUNIT (QUOTE OrderedPrototypes)
                    (AddOnCharacter oTHERs
                                    (QUOTE (VERYSAFESLOT SAFE
                                                         FAST-GET
                                                        FAST-CACHE]
              (FUNCTION (LAMBDA (x)
                 (FindValue (UA~GETVALUE x uSLOT)
                       x uSLOT oTHERs])
(FindDefault)
```

{This scans through the unit's OrderedPrototypes,
returning the first value, V, which is nonNIL.}

```
98~(FindDefault 'Father 'ToPutValue]
DefaultPutValue
```

{Note this is the value stored on TypicalSlot:ToPutValue}

```
100~EDITU(TypicalSlot]
edit
40*f ToPutValue
41*p
... ToPutValue DefaultPutValue
    AfterPutValue DefaultAfterPutValue
    BeforePutValue DefaultBeforePutValue
    BeforeGetValue DefaultBeforeGetValue
    AfterGetValue DefaultAfterGetValue
    ToAddValue DefaultAddValue
    ToDeleteValue DefaultDeleteValue
    ToKillValue DefaultKillValue
    ToSubstValue DefaultSubstValue ...
```

{Note also BeforePutValue and AfterPutValue's values}

```
45*ok
Nothing changed.
TypicalSlot
```

<<<< 26. Strategy: >>>>

What we must do is intercept this FindDefault.
First, we will write the function which actually prints the desired message.
Then create a whole new class of types of slots, which will, by default,
use this chattier put value.

<<<< 27. What gets done when? - system dependent fns >>>>

{First, rewrite the appropriate function, to report Type Errors
Need to see what DefaultPutValue does:}

```
3←PP(DefaultPutValue]
loading from <CSD.RLL>RLL..6

(DefaultPutValue
 [LAMBDA (uNIT sLOT newValue oldValue why sltputter)  **COMMENT**
  {{[OR why (SETQ why (LIST (QUOTE UserCommand]
    (OR oldValue (SETQ oldValue (UA-GETVALUE uNIT sLOT))) }}
  (AND (OR (MEMB (QUOTE Fast-Put) why)
           (APPLY* (GetAccessFn sLOT (QUOTE BeforePutValue))
                  uNIT sLOT oldValue (CONS (QUOTE NewVal) newValue)
                  why))
       [IsOk (SETQ sltputter (GetValue (GetValue sLOT (QUOTE Format)
                                       (QUOTE (VERYSAFESLOT SAFE)))
                               (QUOTE FnForPutting)
                               (QUOTE (VERYSAFESLOT SAFE]
       [SETQ newValue (COND
          ((MustComputep newValue)
           (UA-DELSLOT uNIT sLOT)
           RecomputeMe)
          ((FormattedValuep oldValue)
           (APPLY* (GetValue (ValueFormat oldValue)
                             (QUOTE FnForPutting)
                             (QUOTE (VERYSAFESLOT)))
                   uNIT sLOT newValue oldValue why sltputter))
          ((FormattedValuep newValue)
           (APPLY* (GetValue (ValueFormat newValue)
                             (QUOTE FnForPutting)
                             (QUOTE (VERYSAFESLOT)))
                   uNIT sLOT newValue oldValue why sltputter))
          (T (APPLY* sltputter uNIT sLOT newValue oldValue why]
       (OR (MEMB (QUOTE Fast-Put)
                 why)
           (APPLY* (GetAccessFn sLOT (QUOTE AfterPutValue))
                   uNIT sLOT newValue (CONS (QUOTE OldVal)
                                            oldValue)
                   why))
       newValue])
(DefaultPutValue)
```

{In effect, this first calls (GetAccessSlot slot 'BeforePutValue)
on the arguments.
If that returns nonNIL, it does the put,
and finally (if that also returned nonNIL)
calls (GetAccessSlot slot 'AfterPutValue) on the arguments.}

```
4←GetAccessSlot(Father BeforePutValue]
DefaultBeforePutValue
```

{Recall this appears on TypicalSlot, way above.}

```
5←FindDefault(Father AfterPutValue]
DefaultAfterPutValue

      {ditto}

6←PP(DefaultBeforePutValue]
loading from <CSD.RLL>RLL..6

(DefaultBeforePutValue
{{ [LAMBDA (un sl old modif why)  **COMMENT**
   (COND
     ([OR (MEMB (QUOTE UserCommand)
                why)
          (MEMB (QUOTE UserEdits)
                why)
          (AND (MEMB (QUOTE New-Unit)
                     why)
               (NOT (EQUAL (CDR modif)
                           old]
      (DefaultVerifyValue un sl old modif why))
     (T modif])}}
(DefaultBeforePutValue)

      {Aha - DefaultVerifyValue looks like the function which attempts to verify
       that a value is reasonable.
       (Note it only does this if this is a user edit, ....,
       as it trusts its own puts...)}

7←(DefaultVerifyValue 'Visitor 'Father NIL '(NewVal . HerHusband]
T

8←(DefaultVerifyValue 'Visitor 'Father NIL '(NewVal . HisMother]
NIL

      {Yep, that's the place. So}


                    <<<<      28. Creating a new function    >>>>

9←MOVD(DefaultBeforePutValue ChattyBPV T]

loading from <CSD.RLL>RLL..6
Please enter the Knowledge Base in which to store ChattyBPV: GENLINFO
"ChattyBPV defined using Interpreted Code for DefaultBeforePutValue"

      {(Note I've advised MOVD to be smart - copying the source code rather
       than the compiled code;
       and asking where to store this new function.
       [This is a simple database management facility.]
       I told it to store this function in the list associated
       with the GENLINFO kb.}

10←EDITF(ChattyBPV]
edit
52*←1 2 p
((OR & & &) (DefaultVerifyValue un sl old modif why))
53*←1 p
(DefaultVerifyValue un sl old modif why)
```

```
53*mbd (OR * (PROGN (WRITELNTTY "Unable to put " (CDR modif) " onto " un
      ":" sl " because of a type error!!!") NIL]

55*pp
(OR (DefaultVerifyValue un sl old modif why)
   (PROGN (WRITELNTTY "Unable to put " & " onto " un ":" sl
" because of a type error!!!"))
58*ok

ChattyBPV

                <<<<      29. New class of types of slots - ChattySlots    >>>>

      {First, let me show you NewSubClass:}

93-PP(NewSubClass]
loading from <CSD.RLL>UTIL..7

(NewSubClass
 [LAMBDA (UNAM UOLD whichKB)  **COMMENT**    **COMMENT**
   (NewUnit UNAM UOLD (QUOTE ISubClass)
         whichKB])

(NewSubClass)

      {Recall "ISubClass" was an inheritance' unit we saw long ago.}

94-NewSubClass)
Name: AnyChattySlot
Is a ISubClass of: AnySlot
Please enter the Knowledge Base in which to store AnyChattySlot: GENLINFO

 * Initialized AnyChattySlot *
edit
39*p
(Isa (AnyClassOfObjects)
 AllIsas (AnyCT&U Anything AnyClassOfObjects)
 Prototypes (TypicalClass TypicalCT&U TypicalThing)
 MySlotsNowOrdered (OrderedPrototypes)
 MyCreatedAs (ISubClass &)
 MyTimeOfCreation "18-Aug-80 17:03:13"
 MyCreator "NewUser"
 TotalSoFar 0
 SuperClass (AnySlot)
 MySensibleSlots (Descr Characteristics Prototypes AllSpecs AllGenls AllIsas
            Isa OrderedPrototypes Specializations MySlots MyCreatedAs
            MySensibleSlots MySlotsNowOrdered MyEssentialVirtualSlots
            MyToRenameMe MyToKillMe MyTimeOfCreation MyCreator TotalSoFar --))

39*ok
Verifying slots
AnyChattySlot

     {To make every example of AnyChattySlot print more instructive messages,
      (instead of just returning NIL,)
      we have to intercept FindDefault's search for BeforePutValue.
      This will happen if there is a TypicalChattySlot, which has a
      BeforePutValue value stored.

     So now to create that unit:}
```

```
                <<<<      30. Create a new typical member of AnyChattySlot    >>>>

        {Now to create the TypicalChattySlot unit, and use this new
         ChattyBPV function for its BeforePutValue value}

2←NewTypEx(TypicalChattySlot AnyChattySlot GENLINFO]
 * Initialized TypicalChattySlot *
edit
40*p
(Isa (AnyArchetype)
 TypicalExampleOf AnyChattySlot
 NewPossibleSlots NoEntries
 ... )

41*(n BeforePutValue ChattyBPV]

42*ok
Verifying slots
TypicalChattySlot

      {Now all examples of AnyChattySlot will report such errors. Pf:
       First, the unaltered Husband:}

11←(GetValue 'Husband 'Prototypes]
(TypicalVirtualSlot TypicalUnitFn TypicalStorableFn TypicalProcess TypicalThing
            TypicalCT&U TypicalFunction TypicalSlot)

      {Note Husband's Prototypes, of course, omits TypicalChattySlot}

12←(PutValue 'Visitor 'Husband 'HisMother]
NIL

      {Now move Husband over}

13←(PutValue 'Husband 'Isa '(AnyChattySlot]
(AnyChattySlot)

      {Husband's Prototypes have been rewritten}

14←(GetValue 'Husband 'Prototypes]
(TypicalChattySlot TypicalUnitFn TypicalStorableFn TypicalProcess TypicalThing
            TypicalCT&U TypicalFunction TypicalSlot)
15←redo 11
Unable to put HisMother onto Visitor:Husband because of a type error!!!
NIL

      {Ta daaa!}

16←Examples(AnyChattySlot]
(Husband)

                              <<<<      31. Conclusion    >>>>
      {That's about all for now.}

28←SYSOUT(DEMO]
<3SCRATCH>DEMO.EXE.1

29←LOGOUT]
```

```
It is now 18-Aug-80 17:37:40.
Closing DribbleFile <CSD.IA>TRACE.AUG18.1

@ ; Now in the monitor.
```